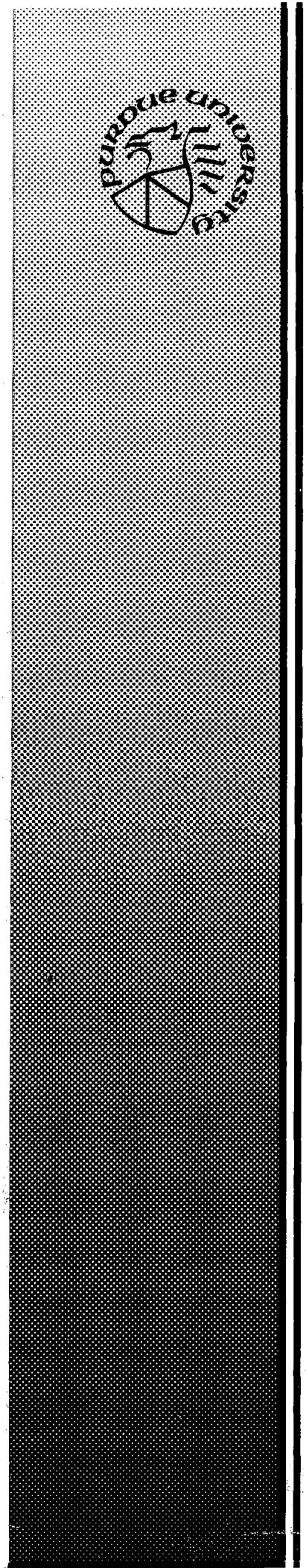
7-1-1988

The PSEIKI Report—Version 2. Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress
Purdue University

A. C. Kak
Purdue University



The PSEIKI Report--Version 2

Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System

K. M. Andress
A. C. Kak

TR-EE 88-9
July 1988

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

The PSEIKI Report -- Version 2

EVIDENCE ACCUMULATION AND FLOW OF CONTROL IN A HIERARCHICAL SPATIAL REASONING SYSTEM

K. M. Andress and A. C. Kak

Robot Vision Lab
School of Electrical Engineering
Purdue University
W. Lafayette, IN 47907

Technical Report TR-EE 88-9
July 1988

Work supported by the Army Night Vision and Electro-Optics Lab

TABLE OF CONTENTS

Page

Page

The PSEIKI Report - Version 2

ABSTRACT

A fundamental goal of computer vision is the development of systems capable of carrying out scene interpretation while taking into account all the available knowledge. In this report, we have focussed on how the interpretation task may be aided by expected-scene information which, in most cases, would not be in registration with the perceived scene.

In this report, we describe PSEIKI, a framework for expectation-driven interpretation of image data. PSEIKI builds abstraction hierarchies in image data using, for cues, supplied abstraction hierarchies in a scene expectation map. Hypothesized abstractions in the image data are geometrically compared with the known abstractions in the expected scene; the metrics used for these comparisons translate into belief values. The Dempster-Shafer formalism is used to accumulate beliefs for the synthesized abstractions in the image data. For accumulating belief values, a computationally efficient variation of Dempster's rule of combination is developed to enable the system to deal with the overwhelming amount of information present in most images. This variation of Dempster's rule allows the reasoning process to be embedded into the abstraction hierarchy by allowing for the propagation of belief values between elements at different levels of abstraction. The system has been implemented as a 2-panel, 5-level blackboard in OPS83. This report also discusses the control aspects of the blackboard, achieved via a distributed monitor using the OPS83 demons and a scheduler. Various knowledge sources for forming groupings in the image data and for labeling such groupings with abstractions from the scene expectation map are also discussed.

CHAPTER 1

INTRODUCTION

This report is an expanded version of the discussion published earlier on PSEIKI, a system for expectation-driven scene interpretation [AndKak88]. In addition to elaborating upon some of the points that were only tersely mentioned in the earlier publication, this report also presents an upgraded version of PSEIKI that can handle both region-based and edge-based symbolic representations of images. The acronym PSEIKI stands for a **P**roduction **S**ystem **E**nvironment for **I**ntegrating **K**nowledge with **I**mages.

PSEIKI can be used for expectation-driven interpretation of vision data in any domain where a good estimate of the expected scene is available. For example, for the navigation of a self-guided munition, PSEIKI could be used to compare an image of the terrain with a map of the terrain; the results produced by PSEIKI could then be used to yield an updated fix on the location of the munition. In more industrial applications, PSEIKI could be used to verify the location and orientation of an object by comparing its 2-D image with a description of the expected scene generated from CAD data. Such verification systems are expected to play an important role for monitoring the progress of assembly robots.

PSEIKI was originally developed for integrating the global map information with vision data to aid navigation for an autonomous mobile robot [KakRob87]. In this problem, the robot's task is to traverse a known network of sidewalks. The primary source of information on the location of the robot is a set of encoders mounted on the wheels; however, due to slippage in the wheels, there is always an uncertainty in the exact location of the robot. PSEIKI's task is to compare the visual image of what the robot sees with the stored map knowledge and obtain an updated position fix on the robot. To simplify this problem of self-location, the cameras on the robot were aimed such that the robot could only see in its immediate vicinity, making for a near-sighted robot.

As a simple illustration of PSEIKI's integration of image and expected scene information in the context of self-location of a mobile robot, consider Fig. 1.1. If panel (a) of this figure is a graphic rendition of an expected scene and panel (b) a depiction of the edges found in the vision data collected for the scene, then PSEIKI would produce an output similar to the one in panel (c), where the labels attached to some of the edges and their corresponding belief values are shown. For example, the label 'right:35%' means that PSEIKI has found the expected-scene edge labeled 'right' in panel (a) to be compatible with the lower right edge in panel (b) with a belief of 35%. In this case, the rest of the belief, 65%, would be apportioned either to this particular label being incorrect or to the system professing ignorance on the subject of assigning a label to this edge in the vision data. The reader might note that the edge labeled 'top:38%' actually corresponds to two edge segments in panel (b). This merger of nearly compatible edges in the vision data is one consequence of various tests PSEIKI makes for internal

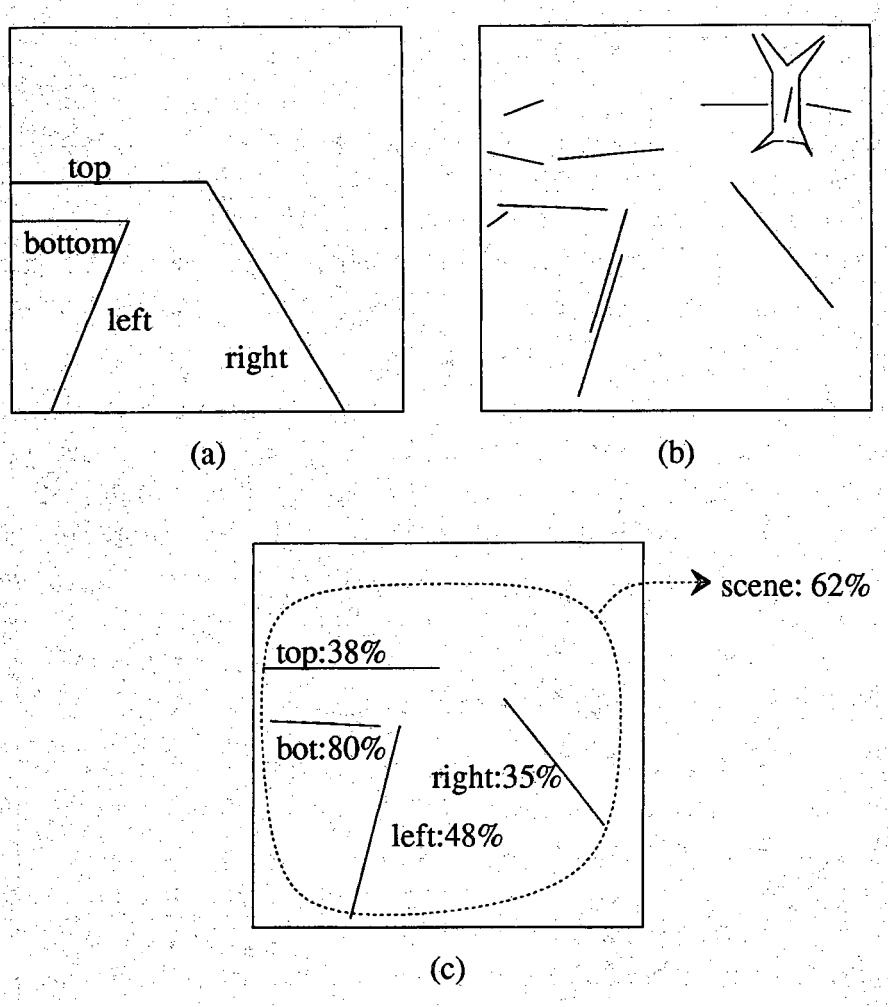


FIGURE 1.1 This figure shows typical images used by PSEIKI. The image in panel (a) shows an example of a graphic rendition of an expected scene with edges labeled. Panel (b) shows a simple example of the output of an edge based preprocessor which PSEIKI would use as input data. Panel (c) shows the final output of PSEIKI, with labeled edges and the labels' belief values in the most plausible interpretation of the scene. The confidence value attached to the overall interpretation of the scene is also shown.

geometric consistencies in the vision data.

The match information generated by PSEIKI is expressed by labeling the image-elements with the identities of the corresponding model-elements; a belief value indicating the confidence of the match found is attached to each label. The Dempster-Shafer theory of evidence is used to accumulate evidence about the certainty of the matches made, a particular advantage of using this formalism being that if a grouping from the image data is too distant or too dissimilar from its correspondent in the expected scene, the system is capable of expressing ignorance for such unlikely associations.* To overcome the exponential explosion usually

*A Bayesian would probably insist one could use low belief values when one is comparing dissimilar or distal groupings in the image data on the one hand and the expected scene on the

associated with the Dempster-Shafer formalism, a computationally efficient variation of Dempster's rule is used to combine evidence about the labels. This variation of Dempster's rule also allows the reasoning process to exploit the hierarchical nature of the integration task. For example, the belief value associated with the top level of the hierarchy is considered to be the confidence in the entire matching process; if this belief value does not exceed a threshold, the matches found are rejected.

Although the original version of PSEIKI, as reported in [AndKak88], was edge based, meaning that both the input image data and the expected scene data had to consist of edge descriptions, the newer version reported here can also handle region-based image inputs directly. However, even when the an image is input into PSEIKI in a region based form, the system still exploits edge level information by treating the boundaries between regions as edges and matching them with edges in the expected scene. We believe that if the edge level information was completely ignored for region based inputs to PSEIKI, the resulting evidence accumulation processes would become weaker, meaning that the system would make weaker assertions about labeling the image regions with entities from the expected scene. As a perhaps poor analogy, given blurry images of, say, a horse and a cow, it may be hard to make a distinction between the two. What we are trying to say is that even in region-based processing, edge level information is not ignored.

PSEIKI groups low-level image-elements into higher level constructs by taking cues from the supplied abstractions in the expected scene. For example, if PSEIKI's low-level preprocessor provides edge information to the system, then PSEIKI would group compatibly labeled, adjacent edges into faces. Once these higher level image-elements are formed, PSEIKI can then match them with high-level model-elements. The following list enumerates the levels of data abstraction present in PSEIKI and describes the data residing on each level.

Level 5:

Scenes -- The entire scene (expected or observed) is represented on this level. The scene is defined as the union of all objects in level 4 of the hierarchy. This level provides a way of labeling multiple objects that otherwise would not be possible. The confidence of the match made on this level is interpreted as the confidence in the entire matching process and is used to determine if the matching process has succeeded.

Level 4:

Objects -- Each element on this level corresponds to a distinct physical object. An object is defined as the union of its boundary faces from level 3.

---

other. We do not dispute that. However, our experience has shown, and as will be illustrated by the discussions in the report, when belief values must be generated from the rather ad hoc measures of geometric compatibility and when such belief values must be normalized for obvious reasons, expressing ignorance by withholding belief becomes a convenient aspect of evidence accumulation -- something that is not allowed in a Bayesian formalism.

Level 3:

Faces -- The elements on this level represent the polygonal faces that form boundary representations of the observable portions of objects. In image data, a face corresponds to a region in the image; in range data, surfaces are stored on this level.

Level 2:

Edges -- These elements represent edges detected in the sensor data; they are used to form the boundaries of the faces in level 3 of the hierarchy.

Level 1:

Vertices -- The vertices are the endpoints of the edges from level 2. They can be expressed either in world or image coordinates depending on the type of data they represent.

Fig. 1.2 shows how a simple scene, a single block, can be broken down hierarchically. Each element in this hierarchy is defined by its parts on lower levels. This figure demonstrates how an object can be defined in terms of its bounding faces and how a face can be defined by the group of edges which form its border.

PSEIKI exploits geometric relationships between data-elements at the above levels of abstraction in the reasoning process. Initial matches between image data and model data are formed by noting geometric relationships between image-elements and model-elements. For example, an image-edge will be matched with the model-edge that comes the closest (in some sense) to lying along the same line in the world coordinate frame. To find the match partner of an image-edge, PSEIKI measures the degree of *collinearity* between the edge and all the model-edges in the vicinity in the world frame; it then chooses as the match partner the model-edge with which the image-edge is most collinear. The belief of the match made then is then made proportional to the degree of collinearity between the two edges.

After the initial matches are made, the extent to which image elements satisfy spatial constraints, dictated by the model information, is used to update the beliefs associated with the assignment of particular model labels to image data. In general, two metrics are required to measure the degree to which image-elements meet these constraints. The two metrics must provide measures of *compatibility* and *incompatibility* between image-elements given the spatial relationships amongst their matched model elements. The compatibility metric provides evidence that an element's label is correct, and, conversely, the incompatibility metric provides evidence that an element's label is incorrect, both from the standpoint of how well the model generated constraints are satisfied. For example, two edges that have the been matched with the same model-edge should lie approximately along the same line. Thus the compatibility metric for edge-elements with the same label, *collinearity(edge1, edge2)*, measures the degree to which the two edges lie along the same line. This collinearity metric is closely related to the measure used to establish initial edge labels, but it is not identical to that measure. The edge-level incompatibility metric, *noncollinearity(edge1, edge2)*, measures the

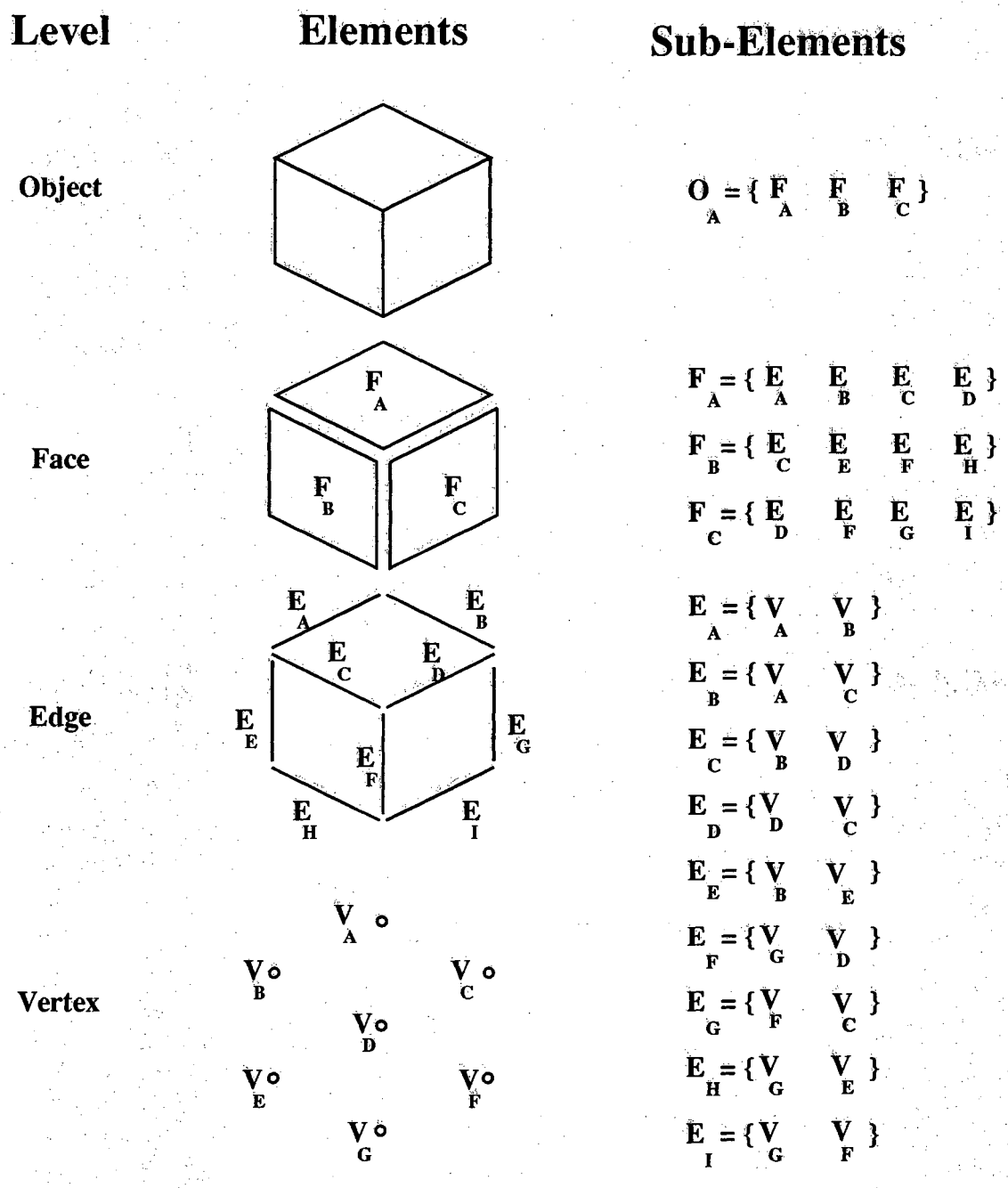


FIGURE 1.2 This figure shows how a simple scene can be broken down hierarchically into objects, faces, edges and vertices.

degree to which two edges do not lie along the same line. Of course, different (in)compatibility metrics must be used at each level of abstraction. For example, the metrics that are used to compute the (in)compatibility between two faces on the data panel are based

on the distance between the two faces' centroids. The metrics used to determine matches and the mechanisms for updating the belief of those matches are discussed in detail in chapters 5 and 6.

An important aspect of evidential reasoning in PSEIKI is the propagation of beliefs up and down the abstraction hierarchy. The propagation of belief values towards the higher abstraction levels is based on the rationale that any evidence confirming a data element's label should also provide evidence that its parent's label is correct. Propagation of beliefs to lower levels is based on the intuitive idea that if, say, a face is mislabeled, then all its constituent edges are also most likely mislabeled.

Although PSEIKI currently is restricted to performing expectation-driven processing on image data, it can also be extended to perform expectation-driven processing on range data. This extendibility stems partially from the independent nature of the blackboard knowledge-sources; the system can be extended by updating or adding a few knowledge sources without worrying about the effect of the extension on the operation of the existing knowledge sources. The system's extendibility also stems from the generic way that PSEIKI treats its data; the data structure that is used to store data-elements can be used to store elements generated by range sensors. Furthermore, the opportunistic nature of blackboard processing can be exploited to tune PSEIKI's flow of control for particular sensors or applications. At the present time, a limitation of PSEIKI is that the expectated scene must be described as an abstraction hierarchy over piecewise-linear edge segments. This implies that any curved boundaries in the scene must be approximated by piecewise linear forms.

In its present configuration, PSEIKI has been implemented in OPS83 as a 2-panel / 5-level blackboard, as shown in Fig. 1.3. The left panel, called the *model panel*, holds the abstraction hierarchy for the expectated scene, and the lower levels of the right panel, called the *data-panel*, are supplied with the image data after it is reduced to a symbolic level. For region-based implementations of PSEIKI, the region level symbolic information from the input image is fed directly to the face level of the data panel. Each level in the blackboard corresponds to one of the levels of data abstraction discussed earlier. Thus each blackboard panel contains the following abstraction levels: scenes, objects, faces, edges and vertices. Each element on the blackboard, except for vertices, is defined by a finite collection of lower-level elements.

PSEIKI has four main knowledge sources (KSs) that it uses to establish correspondences between image-elements and model-elements: *labeler*, *grouper*, *splitter*, and *merger*. The grouper KS determines which data-elements at a given level of the hierarchy should be grouped to form a data-element at a higher level. The merger KS also groups elements; however, its job is to merge multiple elements at a given level and retain the grouped information at the same level. For example, the grouper KS may group together a set of edges into a face; while the merger KS may group together a series of short edge segments into a longer

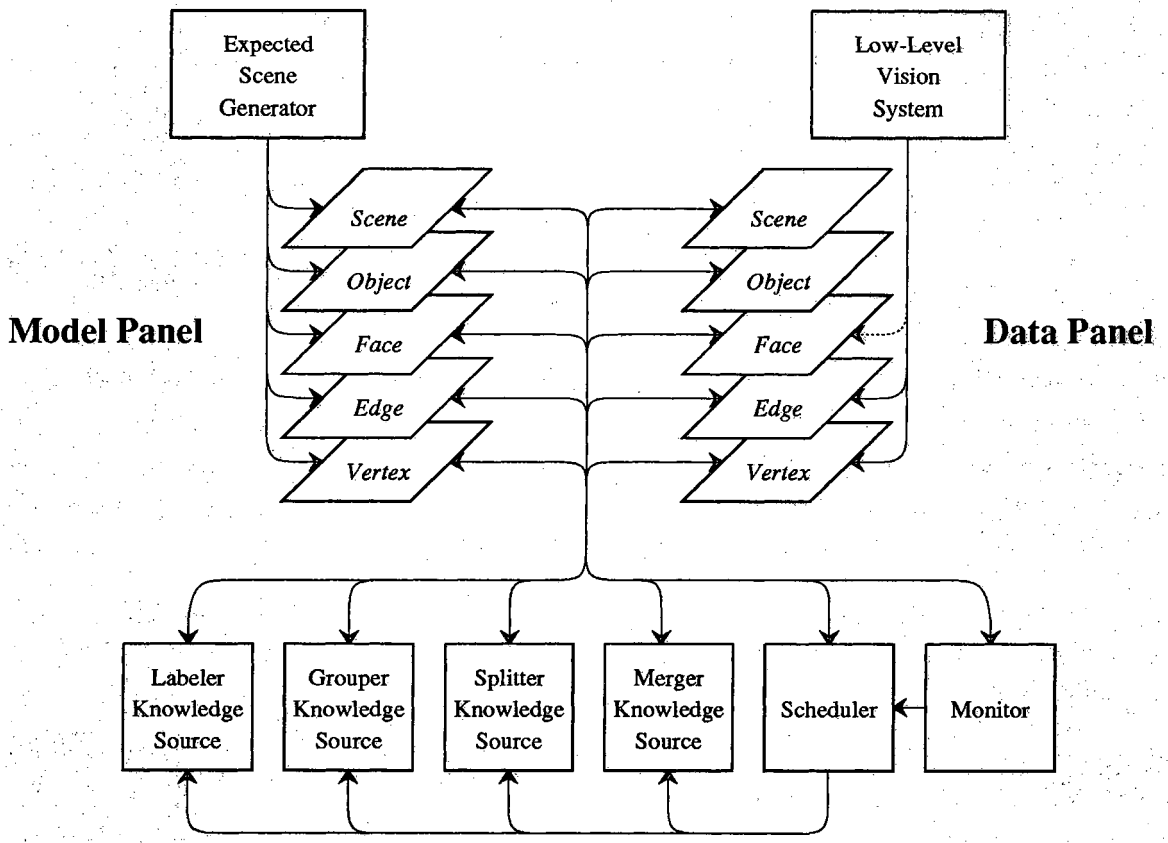


FIGURE 1.3 This figure shows the current configuration of PSEIKI's architecture.

segment, or a set of faces into a single larger face. The splitter KS performs the opposite action of the merger KS; it splits a single element on the blackboard into multiple smaller elements*. The labeler KS has the responsibility of establishing model to data correspondences at all levels of the blackboard, and to accumulate evidence on the validity of those correspondences. Each of these KSs can operate at any level of the blackboard by using level-specific actions.

As was mentioned before, the input image is first preprocessed and then deposited into the lowest two or three levels of the data panel. The type of preprocessing performed by a low-level systems determines the blackboard levels on which the data is deposited. The symbolic information produced by edge based preprocessors is deposited directly at the vertex and edge levels of the data panel. On the other hand, for preprocessors that are capable of

* For those familiar with our earlier publications on PSEIKI, the merger and the splitter KSs in the current implementation are a 'generalization' of the *data-reduction* KS in the earlier version of the system. The data-reduction KS could only operate at the edge level of the blackboard and its function was to merge edge segments into longer edges and to delete short segments. On the other hand, the merger and the splitter KSs can merge and split information at all levels of the blackboard.

producting region type outputs, the additional information is fed directly into the face level of the data panel. This additional input has been depicted by a dashed line in Fig. 1.3. Some low-level systems that can be used to generate input data for PSEIKI are presented in chapter 3. Model data is deposited onto all levels of the blackboard because we assume that perfect knowledge of the expected scene is available.

Work related to PSEIKI will be discussed in the next chapter; a survey of some previous knowledge-based computer vision systems will be presented there. In Chapter 3 we will talk about the type of preprocessing that must be carried out before an image can be fed into PSEIKI; in this chapter, we will also show the data structures used for describing the image elements (the same data structures are used for model elements). Chapter 4 will focus on the generation of expectated scene information and will briefly discuss a couple of CAD systems we have used for this purpose. Chapters 5, 6, 7 and 8 are used to describe PSEIKI in detail. Chapters 5 and 6 are used to present the techniques used in the labeler KS to generate and accumulate evidence for correspondences between the data and the model elements. In particular, chapter 5 is used to describe a hierarchically based evidence accumulation scheme based on the Dempster-Shafer framework; chapter 6 demonstrates how geometric constraints can be used to generate evidence about the matches found between elements. The grouper, splitter and merger KSs are discussed in chapter 7. Chapter 8 is used to discuss the implementation of the blackboard in OPS83; the data structures and the flow of control on the blackboard will be examined. Complexity issues of blackboard processing are addressed in chapter 9. Finally, some preliminary experimental results are presented in chapter 10.

CHAPTER 2

RELATED WORK ON SPATIAL REASONING

It is now realized that for a computer vision system to be able to make scene interpretations in complex environments, the spatial reasoning involved must utilize domain knowledge. Yet, systems that are too domain specific tend to solve problems that are rather narrow in their scope, and given the amount of effort it takes to program such systems, their payoffs tend to be rather limited. In designing PSEIKI, our aim was to create a spatial reasoning tool that would be as domain independent as possible. Clearly, what we have in mind is that a powerful tool like PSEIKI would be used by a higher level, but more domain specific, system for comparing scene expectations with vision data. The higher level system could also guide the inference mechanisms by controlling the various policies used by PSEIKI, such as the policy regarding the priority given to the different KSs, etc. Since our current efforts are focussed on PSEIKI itself, we have not yet addressed how PSEIKI would be embedded in higher level systems that are more domain specific.

In this chapter, we will briefly survey what has been done to date in the development of knowledge based systems for image understanding.

An early model-based image understanding system is described by Brooks in [Bro81]; the task of this system, ACRONYM, consists of finding instances of known objects in the image. To perform object identification, the system first builds an *Observability Graph* that specifies information about objects that could be in the image; generalized cones are used to represent these model objects. The system then builds a *Picture Graph* of the image and identifies instances of objects in the image by matching nodes of the Observability Graph with sets of nodes in the Picture Graph. The objects in the Observability Graph are represented in *slot - filler* structures where any slot that can accept numeric values can also accept algebraic constraints expressed as inequalities. The system then can manipulate these constraints and determine if they are met by properties of objects detected in the image. ACRONYM uses only backward chaining in the matching process and does not incorporate inexact reasoning.

The SIGMA image understanding system for aerial interpretation was first described in [MatHwa85] and later developed in [DavHwa85]. The system represents its object classes hierarchically using frames and uses both forward and backward chaining to arrive at an interpretation of a scene. Furthermore, the system is able to integrate hypotheses about specific objects in the scene. The system does not use uncertain reasoning, but instead is able to control its focus of attention based on the strength of a situation.

Another aerial interpretation system is described by Nagao and Matsuyama [NagMat80]; the system is based on the blackboard architecture and uses multispectral images in the interpretation process. To accomplish the interpretation task, the system first performs a global

survey of the entire image and labels regions without using domain specific knowledge. The *characteristic regions* that it finds, such as water, vegetation, roads, etc., are then used to generate context information for further blackboard processing. This processing consists of a detailed analysis of local areas in the scene using context information provided by the characteristic regions and applying context specific object detection subsystems.

SPAM, a system designed by McKeown, Harvey and McDermott also is an aerial image interpretation system [MckHar85]. The system originally was constructed to interpret airport scenes but has been expanded with a rule generator so it now can interpret scenes from other domains. SPAM uses confidence values to aid in labeling and can manipulate these values based on the consistency of the various labels.

VISIONS (Hanson and Riseman) is another blackboard expert system designed to analyze color images [HanRis78]. The system uses a flexible control scheme, hierarchical scene representation, and a number of knowledge sources to accomplish the scene interpretation task. VISIONS is domain independent, but schemas can be used to tune the system for a particular application.

Nazif and Levine describe an expert system based image segmenter in [NazLev84]; the system was designed to provide a framework that would allow the combination of edge, region and area based segmentation techniques. With these segmentation techniques, the segmenter can split and merge regions, link and break edges and operate on image areas based on features of the elements. The system is rule-based and stores its rules in a global long term memory; the image data undergoing segmentation is operated on in a short term memory. The expert system, which contains a set of metarules, can focus its attention on interesting areas of the image. Many of the processes described in this work are used by PSEIKI to group, split and merge elements on its blackboard.

PSEIKI differs from the above knowledge-based systems in the following three main areas: First, PSEIKI's task differs from those of previous systems. Most of the other systems were designed to find object instances in the image and, through such discoveries, to arrive at a global interpretation of the image. PSEIKI's task is limited to integrating image data with expected scene information -- it generates consistent labels, with associated belief values for the data-elements. Of course, since PSEIKI is limited to matching data-elements, a higher level system is required to make a global interpretation of the scene content.

PSEIKI differs from SPAM and SIGMA, and to a certain extent VISIONS, in not relying on domain-dependent information*. For example, SPAM uses airport design knowledge when

* In the context of this report, a system is called domain dependent when domain-specific knowledge is embedded in various components of the inference engine, such as the rules or the knowledge-sources. PSEIKI is domain independent in this sense; the context information that PSEIKI uses is encoded entirely in the form of the graphic rendition of the expected scene.

interpreting airport scenes. Context-cues also have been used extensively in past computer vision systems. For example, if SIGMA detects a driveway in an image, it then would search for a house and for roads connected to the driveway. Because PSEIKI is provided with a good estimate of the expected scene, it does not have to perform inferences of this type. Although it might be said that context-cues are indispensable for scene interpretation because they make deductions more powerful, their use necessarily introduces some domain dependence. Therefore, it is our philosophy to separate the generation of the mapping from the formation of an overall interpretation of the scene. If the use of context-cues is desired by a system using PSEIKI, then it is up to the higher level system to provide PSEIKI with a graphic rendition of the expect scene incorporating the information contained in the cues.

PSEIKI also differs from previous systems in its method of performing inexact reasoning. Many systems, including ACRONYM, SIGMA and the system by Nazif and Levine use no uncertain reasoning in the image interpretation process. Because of the overwhelming amount of data in an image, most of the inexact reasoning schemes used in the past have employed simple combination schemes in order to keep from becoming bogged down in certainty value computations. On the other hand, inexact reasoning in PSEIKI is based on the Dempster-Shafer formalism in a tangled hierarchical space. The use of a hierarchy curtails the number of uncertainty calculations and is made possible by the use of the blackboard architecture.

CHAPTER 3

PREPROCESSING OF INPUT VISION DATA

The image to be interpreted must first be converted into a symbolic form before it can be deposited on the lowest two or three levels of the data panel of the blackboard. This chapter will focus on the preprocessing steps that we use for this conversion to symbolic form. In this chapter, we will also describe the the format in which PSEIKI expects to see the input symbolic data. The same format is also used to pass expected scene information to PSEIKI, more on that in Chapter 4.

The chapter will describe two image segmenters, one is edge-based and the other region-based. The former is used for generating edge-based symbolic descriptions of the input image, and the latter for region-based symbolic descriptions. The two segmenters described here are presented only as examples of systems that can generate input data; because they both use well known techniques, they will not be described in any great detail. Furthermore, no claim of optimality for any of the presented systems is made. In fact, for PSEIKI to be a truly general expectation-driven vision system, it should be robust enough to overcome any peculiarities of these or most other low-level preprocessors. Thus, if improved low-level preprocessing techniques become available in the future, PSEIKI should be general enough to use the segmentation produced by the new preprocessors.

3.1. Format of Input Data

PSEIKI expects to see its input data as an ASCII text file with each line corresponding to a separate data element, as shown in Fig. 3.1. The fields used in the data files are self-explanatory. The first field on a line following the '+' specifies the level of the blackboard onto which the element is deposited. All other fields are specified by *keyword - data* pairs; the data part of some fields can hold multiple values. For example, the data part of the children field can specify that an element has more than one child. The **id** field is used to specify a unique identification number for a data element; each element on the blackboard is referenced via its id number. The element's **children** field specifies the sub-elements that are used to build it; for example, an edge has two children -- its end vertices. If an element is a vertex, its location may be specified in one of two ways. If the vertex is on an image plane, its location must be specified via the **row** and **col** fields. However, if the vertex is to be specified in three-space, the **coordinate** field is used to specify its location in world coordinates; the data part of this field holds three values -- the x, y and z values of its location. Any text appearing after a semicolon is considered to be a comment and is ignored. Besides the fields shown in Fig. 3.1, there are a number of optional fields that the low-level systems can use to provide additional

```
+ object  id 1     children  2  3  4          ; object A

+ face    id 2     children  5  6  7  8       ; face A
+ face    id 3     children  7  9 10 12       ; face B
+ face    id 4     children  8 10 11 13       ; face C

+ edge    id 5     children 14 15             ; edge A
+ edge    id 6     children 14 16             ; edge B
+ edge    id 7     children 15 17             ; edge C
+ edge    id 8     children 16 17             ; edge D
+ edge    id 9     children 15 18             ; edge E
+ edge    id 10    children 17 20             ; edge F
+ edge    id 11    children 16 19             ; edge G
+ edge    id 12    children 18 20             ; edge H
+ edge    id 13    children 19 20             ; edge I

+ vertex  id 14    coordinates 1.0 1.0 1.0    ; vertex A
+ vertex  id 15    coordinates 1.0 0.0 1.0    ; vertex B
+ vertex  id 16    coordinates 0.0 1.0 1.0    ; vertex C
+ vertex  id 17    coordinates 0.0 0.0 1.0    ; vertex D
+ vertex  id 18    coordinates 0.0 1.0 0.0    ; vertex E
+ vertex  id 19    coordinates 1.0 0.0 0.0    ; vertex F
+ vertex  id 20    coordinates 0.0 0.0 0.0    ; vertex G
```

FIGURE 3.1 Sample data file demonstrating PSEIKI's input data file format

information to PSEIKI. The **value** field can be used to provide PSEIKI with a level specific value; for example, this field can be used to indicate an edge's average strength or a region's average grey level. Likewise, the **size** field can provide PSEIKI with level specific size information (e.g. region area, edge length, degree of a vertex).

The input data presented by the edge-based preprocessor is deposited on the edge and vertex levels, in this description the vertices may be described by either the image based coordinates or their 3-D world coordinates. On the other hand, in addition to the edge and vertex level information, the region-based preprocessor also feeds information at the face level. The data on the face level represent the regions extracted by the segmenter; the borders between these regions would be represented as edges, and would be described at the edge level. Finally, the vertices associated with the edges would be input at the vertex level.

3.2. An Edge Based Image Preprocessor For PSEIKI

Edge detection is a common technique used in image segmentation and other low-level image processing [RosKak82], [BalBro82]. However, the most common edge detection process, which consists of thresholding the output of a gradient type window operator, is incapable of generating input data directly for PSEIKI. This is due to the difficulty encountered when converting thick edges produced by this process to the symbolic form required by PSEIKI. Although iterative methods are available to reduce the widths of these edges, they are prohibitively time-consuming [RosKak82], [Ebe76], [BalBro82]. Ridge-tracking is another method that can be used for edge detection [WatArv87]. A variation of the ridge-tracking algorithm described in [Kim88], which lends itself to the conversion of edges into a form usable by PSEIKI, will be described in this section. A modification of the original algorithm was necessary due to PSEIKI's requirement that all of its input data be represented symbolically. The original algorithm's inability to find edge intersections also has been corrected in PSEIKI's preprocessor. There are a number of steps to the modified segmentation process.

1) First, a window-based gradient operator is applied to the image; the Sobel operator is used in the current system [RosKak82]. Since the ridge-tracking algorithm uses only gradient magnitude information, the direction of the gradient is not computed.

2) After the gradient operator is applied to the image, every pixel above a user-specified threshold is stored in a list; this list of pixels is called the *threshold list*. Since the system only works on pixels in this list (usually between 5% and 10% of the total number of pixels), the required amount of work is drastically reduced.

3) To reduce the algorithm's noise sensitivity, all pixels in the threshold list are averaged with their eight closest neighbors.

4) The next step in the process consists of finding all edge endpoints; eventually, these pixels correspond to vertices on PSEIKI's blackboard. To find these elements, the notion of the *degree of one dimensional maximum* (DODM) is used. Each pixel has four pairs of neighbors -- horizontal neighbors, vertical neighbors, and neighbors in two diagonal directions. The DODM of a pixel is the number of pairs of neighbors in which both neighbors have lower values than the pixel itself. Fig. 3.2 demonstrates this concept; the DODM for the center pixel, "C", is defined to be the number of cases in which it is larger than both of its two neighbor pixels, "N". The center pixel of the image neighborhood shown in Fig. 3.3 has DODM 2 since it is larger than its four neighbors in the horizontal and vertical directions. All pixels in the threshold-list with DODM of three or four are considered to be edge endpoints.

5) It is in the next step in segmentation that the ridge-tracking process actually occurs. Two image structures are used to aid in this ridge-tracking process; these image structures are called the *edge* and *mark* arrays. The edge array is used to record the pixels that have

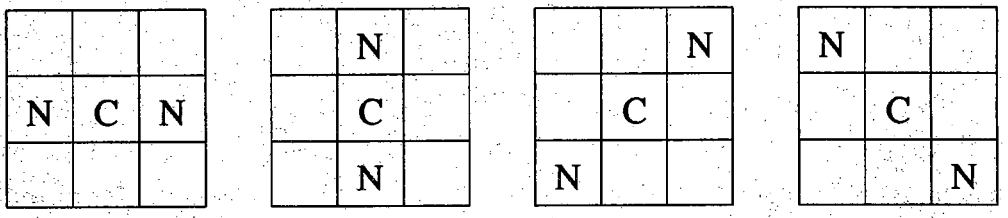


FIGURE 3.2 This figure demonstrates the concept of the Degree of one Dimensional Maxima (DODM). The DODM for the center pixels is defined to be the number of cases (1-4) in which the center pixel "C" is larger than both adjacent pixels "N" along a line.

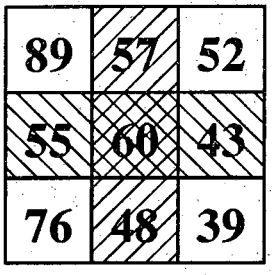


FIGURE 3.3 The DODM of this example image neighborhood is 2 because the center pixel has a larger value than its horizontal and vertical neighbors.

been determined to be endpoints or parts of an edge. If the value of a pixel is nonzero in the mark array, then the pixel is said to be marked and the tracker will not follow the edge onto that pixel. This technique is used to keep the tracker from backtracking onto pixels recently determined to be part of the edge. Another concept that is used in the tracking process is called the *current(i) pixel*; this is the ridge pixel that was determined, at time i, to be part of the edge. The tracking process is described below.

5a) Let $i = 0$. Obtain an endpoint vertex found in step 4 of the process and designate this as the current(0) pixel. In the edge array, label this pixel as an endpoint and mark this pixel in the mark array (by setting the value of the pixel in the mark array to nonzero).

5b) In the edge array, label the current(i) pixel (if $i \neq 0$) as an edge pixel and let $i = i + 1$.

5c) Choose the current(i) pixel in the following manner: If there is an unmarked endpoint or edge pixel adjacent to the current($i - 1$) pixel in the edge array, choose this unmarked pixel as the current(i) pixel, designate it as an endpoint in the edge array, and stop the tracking process. Otherwise, find the next pixel in the edge by finding the largest unmarked pixel which is adjacent to the current($i - 1$) pixel and has DODM ≥ 2. Label this pixel as current(i), add it to a list of pixels that denote the edge, and designate it as an

edge in the edge array. If there exists no pixel fitting this description, then the edge "died;" designate the current(i – 1) pixel as an endpoint and stop the tracking process.

5d) Unmark the current(i – 2) pixel and its eight neighbors.

5e) Mark the current(i – 1) pixel and its eight neighbors.

5f) Go to step (5b).

The original algorithm never unmarked pixels after they were marked; this prevented the system from finding junctions between edges. By unmarking pixels when there is no possibility of the ridge-tracker backtracking onto freshly labeled edge pixels, these vertex pixels can be found. If the number of pixels in an edge is less than a user specified threshold, then the list is deleted and all pixels in the edge matrix are reset to their original state.

A few iterations of the tracker at step (5e) are shown in Fig. 3.4 to demonstrate how the tracking algorithm works. In this illustration, the pixels in boldface have been labeled as belonging to the edge. The shading denotes pixels that have been marked on the current iteration of the tracking algorithm.

6) The final step of the segmentation process is the fitting of piecewise-linear segments to the lists of edge pixels. This step is based on a process described in [DudHar73] and also used in [NavBab80]. This step also requires a user-specified parameter -- the maximum fitting error, E_{max}. In this process, a line, called the *model* line, is drawn between the two endpoints of an edge; then the edge pixels are followed (by traversing the list of edge pixels) and the distance between the individual pixels in the edge and the model line is computed. If the distance between every pixel and the line is less then E_{max}, then the edge can be represented by the model line. However, if any pixels are greater than E_{max} away from the model line, then the pixel that is the farthest from the model line is considered to be a new endpoint and the line fitting algorithm is called recursively (once for each edge between the new endpoint and the old endpoints). The line fitting process is shown in Fig 3.5; in this example, the line-fitting process breaks the line into two piecewise linear segments.

The segmentation process, including the intermediate steps, is shown in Figs 3.6 and 3.7. Fig. 3.6 demonstrates the process when applied to an image typical of those taken by a mobile robot with downward pointing cameras. Fig. 3.7 demonstrates the process when applied to an industrial scene. In each of these two figures, panel (a) shows the original image; panel (b) shows the magnitude of the gradient as found by the Sobel operator, and panel (c) shows the edges that were traced by the ridge-tracking algorithm. Panel (d) shows the final output of the segmenter after it has converted the edges in panel (c) into piecewise-linear segments.

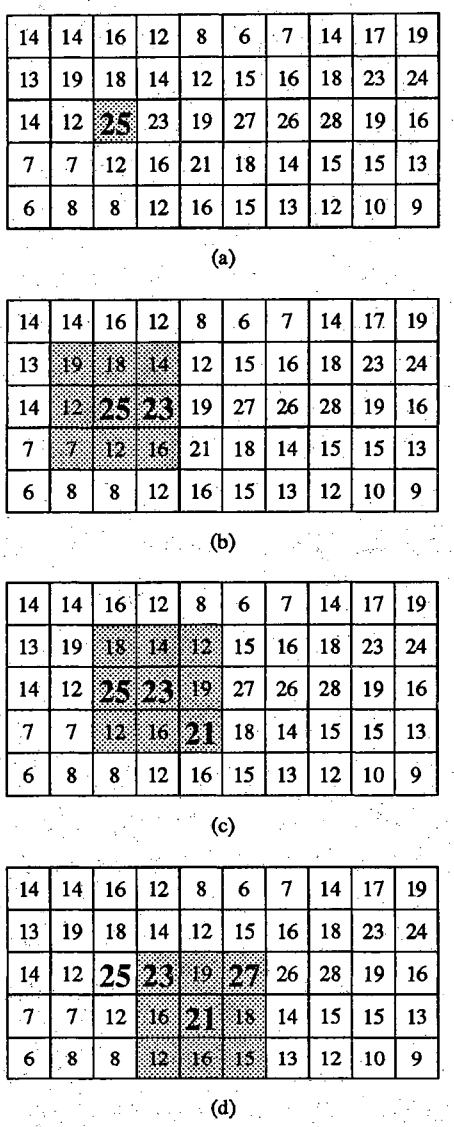


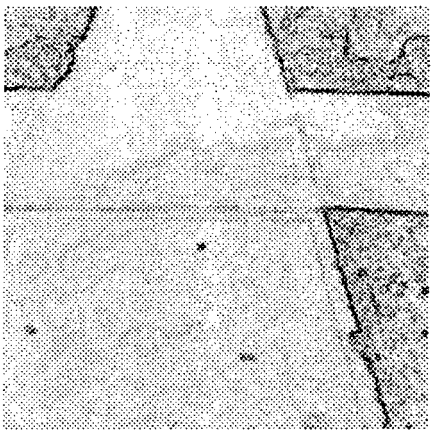
FIGURE 3.4 This figure demonstrates the marking of pixels in the ridge-tracking algorithm. The boldface pixels represent edge pixels and the shaded pixels are marked.

This process is fairly efficient due to the use of linked lists to represent the edges. The segmenter was applied to a set of 512×480 test images; the system was able to segment an image (perform the Sobel operation, threshold, smooth, ridge-track and convert to symbolic form) in an average of 45 seconds on a lightly loaded SUN/3.

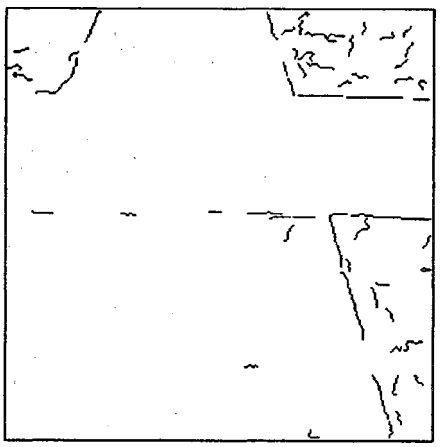
3.3. A Region Based Image Preprocessor For PSEIKI

As was mentioned before, PSEIKI can be driven in two modes: in the first mode the input image is first reduced to an edge-based description and the resulting description used for deriving abstraction hierarchies; in the second, mode abstraction hierarchies are built on top of

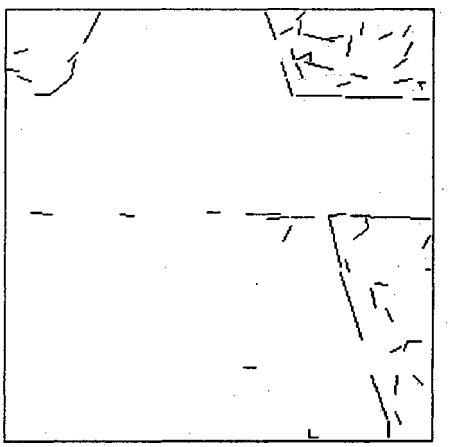
(a) original image



(b) image after applying sobel operator

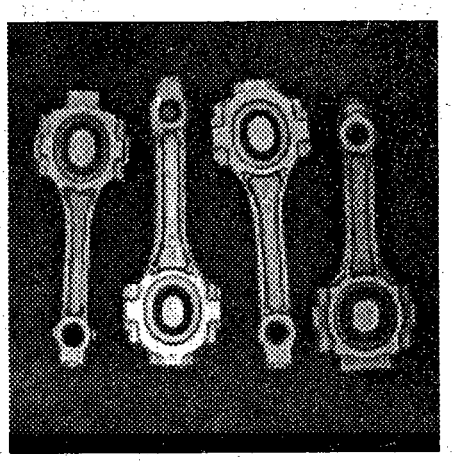


(c) edge found by ridge follower

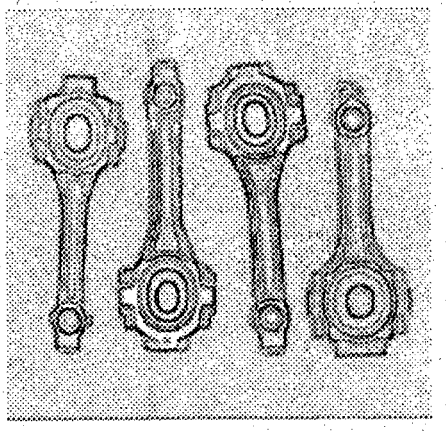


(d) edges after conversion to piecewise-linear segments

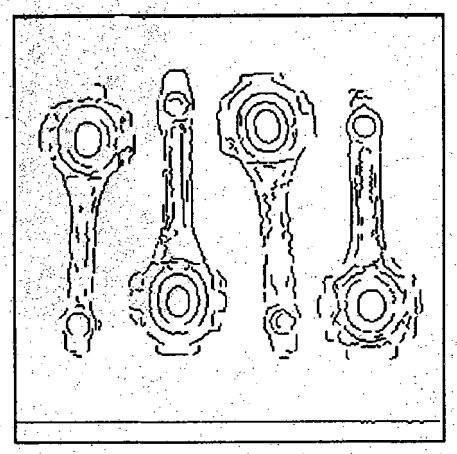
FIGURE 3.6 This figure shows the intermediate and final output of the edge-based preprocessor when applied to an image typical of those gathered by a mobile robot with downward pointing cameras.



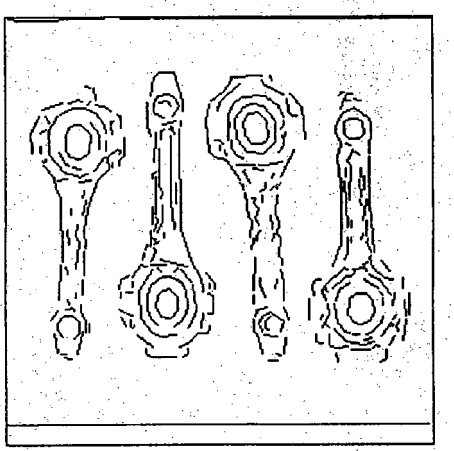
(a) original image



(b) image after applying sobel operator



(c) edge found by ridge follower



(d) edges after conversion to piecewise-linear segments

FIGURE 3.7 This figure shows the intermediate and final output of the edge-based preprocessor when applied to a typical image of an industrial scene.

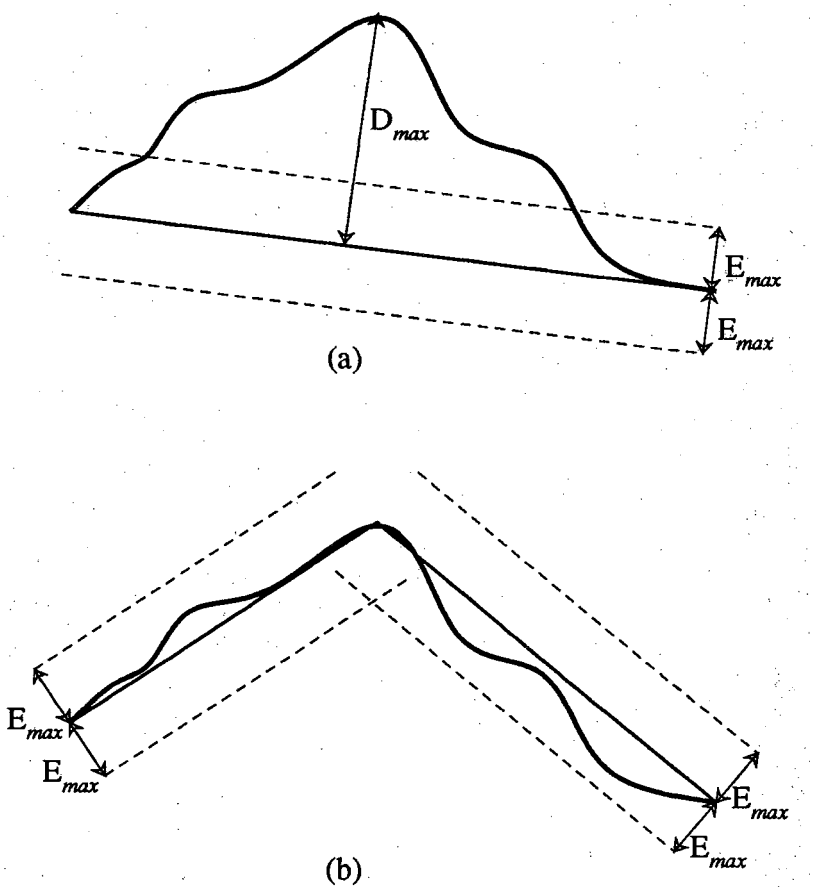


FIGURE 3.5 This figure demonstrates how a sample edge could be broken into two piecewise-linear segments by the line fitting algorithm. Since the edge falls outside the E_{max} boundaries in (a), the line is split into two in (b) where the edge lies within the E_{max} boundaries.

region-based descriptions of the input image. For the second mode of operation, we use a segmenter based on region growing ideas first advanced in [BriFen70] and later further developed by Horowitz and Pavlidis in [HorPav76]. Our implementation differs from that described in [HorPav76] in that we use the *quadtree* data structure that has become rather popular since the original algorithm was published. The quadtree data structure, a well known tool for representing binary images [Sam84a, Sam84b], has been extended in this application to represent greyscale images. There are a number of steps that the region growing process uses to generate the final segmented image.

1) The segmenter's first step is to break the image into a data structure called a *greyscale quadtree*. A greyscale quadtree is a simple extension of the binary quadtree in which every leaf is maximal and satisfies a constraint (a leaf is maximal if it is not part of a larger leaf that satisfies the constraint). In this segmenter, a group of pixels is allowed to be grouped into a leaf of a quadtree if

$$\left\{ \max_{x,\,y} f(x, y) - \min_{x,\,y} f(x, y) \right\} \leq 2\varepsilon \qquad (3.1)$$

where f(x, y) denotes the brightness function of the image and x, y are allowed to range over the entire leaf; epsilon is a user-supplied parameter. In the original algorithm, this process required an iterative split-and-merge procedure. However, with the use of the Morton matrix [Mor66], [Sam84a] the quadtree can be built without any iterations. By visiting the pixels in the order defined by the Morton matrix, the building of a leaf can be postponed until it is certain that no larger leaf node satisfying constraint (3.1) is possible. Fig. 3.8 shows an example of an 8×8 Morton matrix.

1	2	5	6	17	18	21	22
3	4	7	8	19	20	23	24
9	10	13	14	25	26	29	30
11	12	15	16	27	28	31	32
33	34	37	38	49	50	53	54
35	36	39	40	51	52	55	56
41	42	45	46	57	58	61	62
43	44	47	48	59	60	63	64

FIGURE 3.8 An example of an 8 by 8 Morton Matrix.

Note that the Morton matrix does not have to be stored explicitly to guide the traversal of the image in the order that it prescribes. An image can be traversed in the correct order by recursively visiting the four quadrants of the image in the following order: northwest, northeast, southwest and southeast.

2) The segmenter's second step is to merge adjacent quadtree leaves into regions. Adjacent leaves are merged into a region only if the region formed also satisfies constraint (3.1). Regions of the image are represented using the tree based UNION-FIND data structure described in [AhoHop74].

3) The third step in the process is the merging of adjacent regions whose average greyscale values differ by less then a user specified threshold.

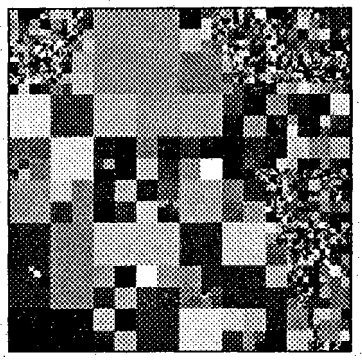
4) At the end of these processes there may exist very small regions that should be eliminated; for example, many of these regions are generated by shot noise and are only a single pixel large. Each small region is merged with the neighboring region whose average grey level is closest to its own.

5) The segmenter's final step is to convert the segmented image into a format usable by PSEIKI. This is accomplished by first finding all the borders between regions; these border-elements are then converted into piecewise-liner segments using the process discussed in the previous section. The endpoint pixels are output as vertex-elements for the blackboard; likewise, the borders and regions are output as edges and faces respectively.

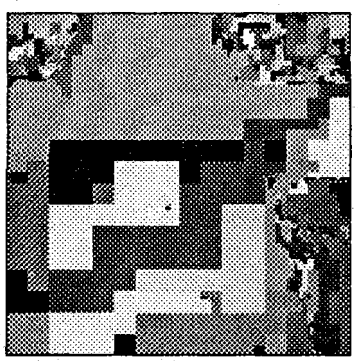
The segmentation process, including the intermediate steps, is shown in Figs. 3.9 and 3.10. These two figures demonstrate the region growing process when it is applied to the sample images described in the previous section. In each of these two figures, panel (a) shows the original image; panel (b) shows the quadtree leaves generated by step 1 (the grey-levels in these images are arbitrarily generated and are used to help the reader distinguish between adjacent regions). Panels (c), (d) and (e) show the regions after steps 2, 3 and 4 are used to generate and merge regions. Panel (f) shows the region borders after they are converted into piecewise-linear segments.

This segmenter is slightly less efficient than the edge-based process; however, the system was able to segment 512×480 images in less than two minutes on a lightly loaded SUN/3.

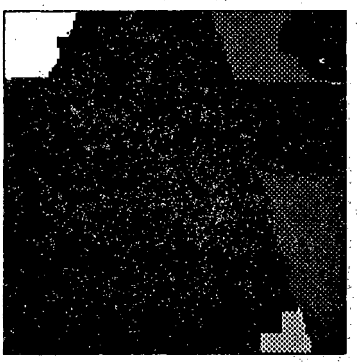
(a) original image



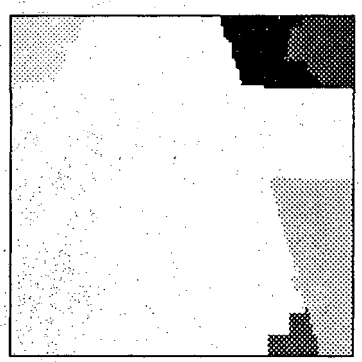
(b) greyscale quadtree



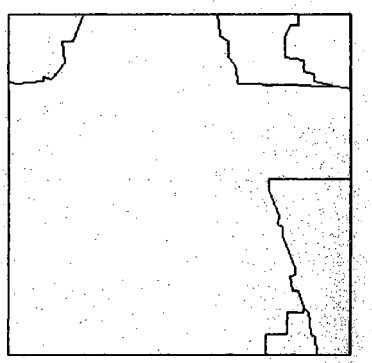
(c) regions after max-min merging



(d) regions after merging based on averages

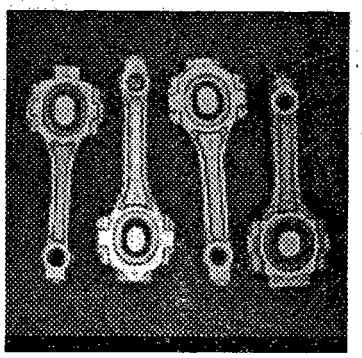


(e) final result after merging small regions

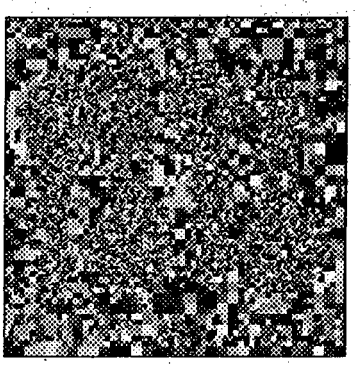


(f) symbolic output

FIGURE 3.9 This figure shows the intermediate and final output of the region-based preprocessor when applied to an image typical of those gathered by a mobile robot with downward pointing cameras.

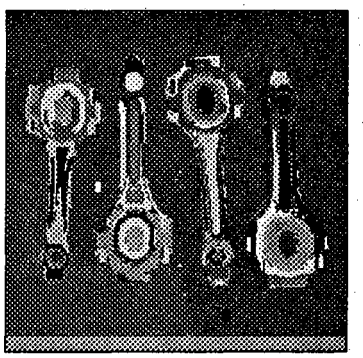


(a) original image

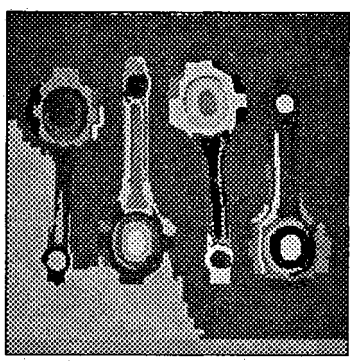


(b) greyscale quadtree

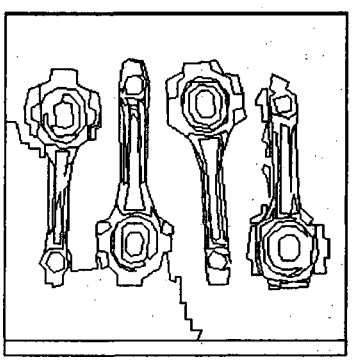
(c) regions after max-min merging



(d) regions after merging based on averages



(e) final result after merging small regions



(f) symbolic output

FIGURE 3.10 This figure shows the intermediate and final output of the region-based preprocessor when applied to a typical image of an industrial scene.

CHAPTER 4

EXPECTED SCENE GENERATION

Computer graphics systems and CAD systems are two obvious methods of generating PSEIKI's expected scene information; this chapter will present two systems used to generate model information for PSEIKI. A computer graphics interface is used to generate the expected scene information for mobile robotic applications, while a solid modeling package is used for more industrial domains. Any modeling tool that is used for expected scene generation must possess the capability for hidden line removal. Also, the modeling tool must output its information in the same format that was described in Section 3.1. Note that while the symbolic information that is input on the data panel has at most two or three levels initially, the expected scene has to be described as a hierarchy containing descriptions at all levels.

4.1. Expected Scene Generation for Sidewalk Navigation Applications

For sidewalk-navigation applications, a simple 2D graphics program is used to generate PSEIKI's expected scene information from stored sidewalk maps. In this system, the sidewalk maps are stored in a graph data structure. The links in this graph represent straight sections of the sidewalk and nodes represent the endpoints of the straight sections. Associated with each node is an (x, y) pair designating the coordinates of the sidewalk junction corresponding to the node; thus, the centerline of a straight section of sidewalk is the line that connects the coordinates of its two junction nodes. Associated with each link of the graph is a numerical value that specifies the width of the corresponding segment of the sidewalk. This is enough information to completely specify a sidewalk map.

Fig. 4.1 illustrates the steps involved in the generation of a symbolic description of the expected scene from the graph data structure. The first step involved in generating the expected scene information is the extraction of the edges of the sidewalk from the graph data structure. It is a trivial task to determine the lines defining the edges of a straight section of the sidewalk because both the section's width and its centerline are known. A more difficult problem is encountered when trying to determine the location of the vertices corresponding to the intersection points of the edges of the sidewalk. These are determined by the following algorithm. First, we associate four vertices with each link in the graph corresponding to the two endpoints of each of its two edges. For example, we associate the vertices P, Q, R and S with the link AB as shown in Fig. 4.2. Vertices P and Q are obtained by analyzing node B, whereas vertices R and S are obtained by analyzing node A. Consider node B first. The graph is searched for all the links that meet at B; the angle that each link subtends with the link AB, measured in a counterclockwise fashion, is then calculated. We then retain only those links

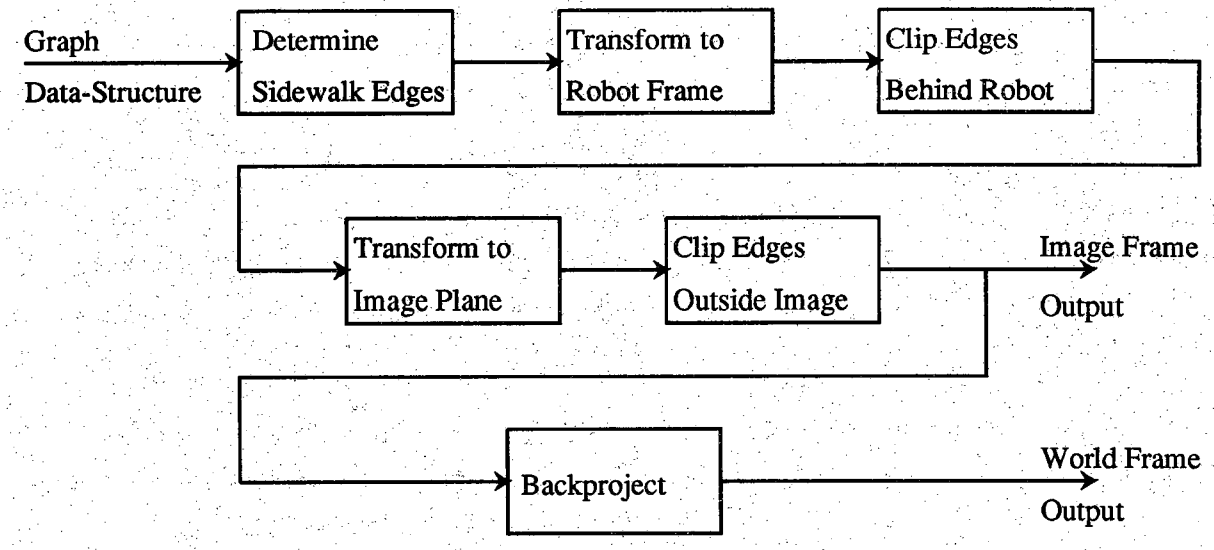


FIGURE 4.1 This figure shows a block diagaram of the processes used to generate PSEIKI's expected scene information in a mobile robotic context.

that correspond to the minimum and maximum of these angles. As shown in the figure, links BC and BE correspond to the minimum and maximum angles there, respectively. Now it is rather simple matter to compute the location of the two vertices, P and Q, that correspond to node B of link AB; for example, the computation of the location of vertex P can be found by solving the equations of the straight lines corresponding to the edges SP and PT. Similarly, the location of vertices R and S can be computed by analyzing node A. Note that at node A, where there is only a bend in the sidewalk, as opposed to a junction, the minimum and the maximum angles correspond to the same link, that is the link AF. The pseudo code in Fig. 4.3 presents the algorithm more formally. The reader should note that this procedure will yield each vertex, such as point P in the figure, twice. In this example, the vertex corresponding to point P will be generated when node B is considered as belonging to link AB, and then again when the same node is considered as belonging to link BC. This duplication at the vertex level of the symbolic description is easily eliminated by comparing vertices and dropping one when two are found to be nearly identical in terms of their coordinates.

After a symbolic description of the edges in the sidewalk map has been extracted from the graph data structure, a "spotlight" function is applied to the description to delete all those edges that can not be seen from the robot's hypothesized location and orientation. To implement the spotlight function, we first generate two homogeneous transformation matrices, one that takes a world point into the robot base coordinate frame and the other that takes a point from the robot base coordinate frame into the camera image plane. The first matrix, which is derived from knowledge of robot's location and orientation, is used to transform end points of edges, such as point P for edge PS in Fig. 4.2, from the world frame into a robot base

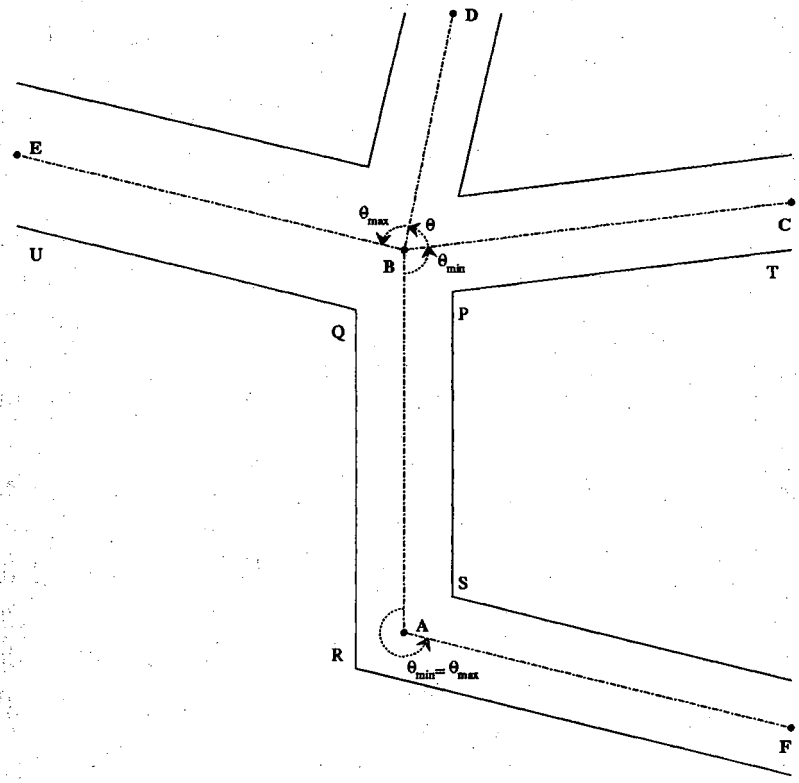


FIGURE 4.2 This figure shows a part of a sidewalk map; it shows the process used to generate a sidewalk's edges from a graph description.

coordinate frame. A clipping operator is applied to the transformed data to delete all those edges that are behind the robot. The middle panel of Fig. 4.4 illustrates the edges from the left panel that would remain after this clipping operation is applied. Now the second transformation matrix, which is derived from camera calibration parameters, is applied; this transformation is used to project the clipped edges onto the camera image plane. A second clipping algorithm is now applied to delete the edges and parts of the edges that fall outside the boundaries of the image. The edges from the middle panel of Fig. 4.4 that are not deleted by the final clipping operation are shown in the right panel. Note that the edges of the sidewalk are still described symbolically at this point; that is, they have not been converted into image form.

If the expected scene is to be expressed in world coordinates, the clipped edges are then back-projected into the world coordinate frame. This vertex level and edge level information is finally output in the format described in section 3.1. The project/clip/back-project process just described has the desired affect of deleting all of the edges that are not visible from the robot's hypothesized location and orientation. If the expected scene is to be expressed in image coordinates, then the system outputs the edges in the appropriate format without back-projecting them. Although it would be possible to implement the world coordinate spotlight function via a simple clipping operation performed in the world coordinate frame, using the

```
get_both_edges(LINK, right_start, right_end, left_start, left_end) {
        get_vertices(LINK, start_node(LINK), left_start, right_start)
        get_vertices(LINK, end_node(LINK), right_end, left_end)
}

get_vertices(LINK, NODE, right_vertex, left_vertex) {
        for each link in the graph not equal to LINK {
                if (one of the link's nodes is equal to NODE)
                        add the link to the set of intersecting links
        }

        sort the intersecting links on the basis of the angle between them and LINK

        min_link = link with minimum angle
        max_link = link with maximum angle

        right_vertex = intersect(edge(LINK, right), edge(min_link, right))
        left_vertex = intersect(edge(LINK, left), edge(max_link, left))
}
```

FIGURE 4.3 This figure shows the pseudo code for algorithm used to determine the location of the vertices of a section of the sidewalk.

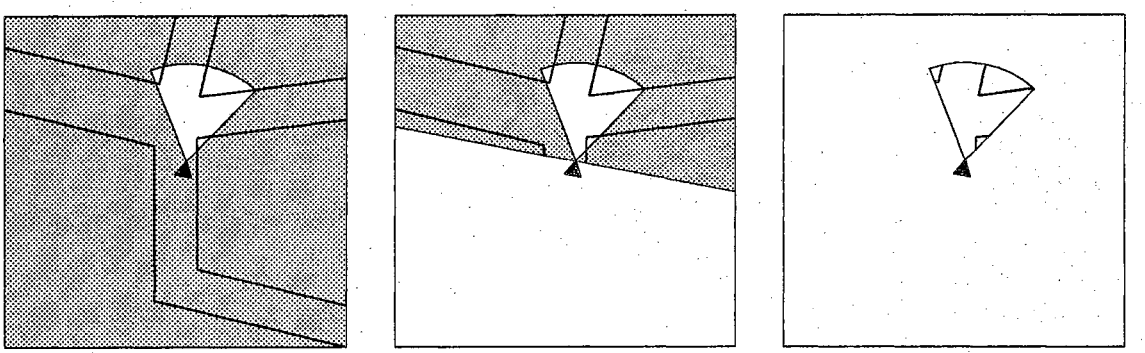


FIGURE 4.4 This figure shows how the spotlight function is used to delete from the expected scene all edges that can not be seen from the robot's hypothesized location and orientation. In the leftmost panel, the triangle shows the expected location and orientation of the robot and the unshaded area shows the region of the ground visible to the robot's downward slanted cameras. The center panel shows the clipping of the edges behind the robot. The rightmost panel shows the edges remaining after the image-coordinate clipping is performed.

project/clip/back-project algorithm allows us to use a single spotlight function for both world coordinate and image coordinate output.

After the low-level information is generated by the graphics system, the model information on the face level, the object level and the scene level is hand entered by editing the output file. On the face level, each connected section of sidewalk and each connected section of the ground is considered to be a separate face. These faces are hand grouped onto the single object in the scene. To help the operator enter this upper-level information, an image of the expected scene, with the grey values of each edge indicating its symbolic id number, is displayed at the same time the low-level symbolic output is generated. Generating this image is trivial because the spotlight function projects the sidewalk's edges into the image coordinate plane. Hand entering the upper-level information is usually not difficult because the sharp down-look angle of the camera limits the complexity of the expected scenes.

As an example of the processing performed by this graphics system, consider the following figures; Fig. 4.5 shows a simple sidewalk map to be used in this example.

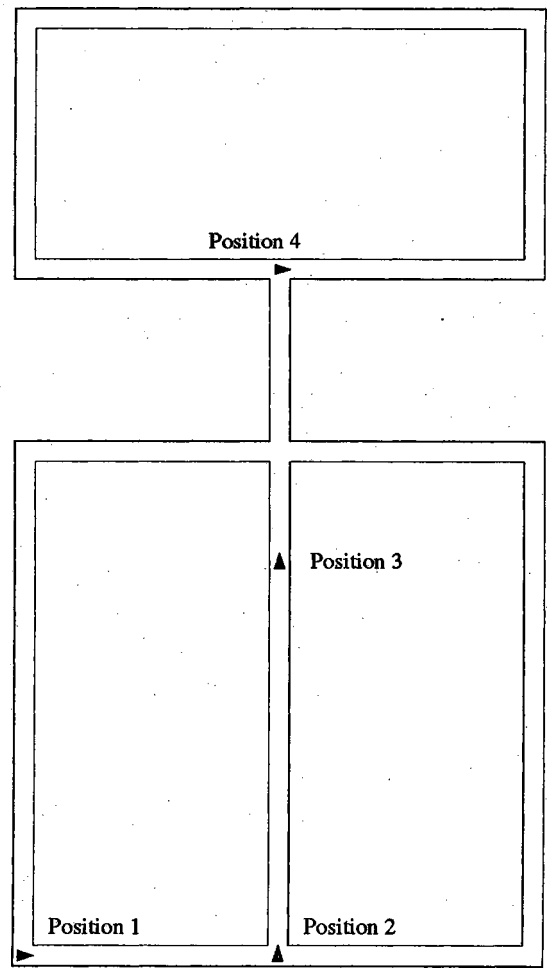


FIGURE 4.5 This figure shows the sidewalk map used to generate the expected scene images of figure 4.6. The robot's position for each of the four expected scenes is indicated in the drawing.

Fig. 4.6 shows a sequence of expected-scene images that the system would produce for a mobile robot traveling to the middle sidewalk section of the map, turning up that section and then turning right.

4.2. Expected Scene Generation for Industrial Applications

A generic solid modeling system is used to generate PSEIKI's expected-scene data for industrial applications. Solid modeling techniques have gained great popularity in the past decade for representing geometric objects in a complete and unambiguous fashion. Constructive solid geometry (CSG) and boundary-representation (B-rep) are the two most popular solid-modeling techniques. In this section, we will first highlight the principles used in CSG based modeling and show an example of an object constructed using CSG principles. We will then describe the TWIN B-rep modeling system and describe how the system is used to generate PSEIKI's expected scene information.

Solid objects are created in CSG systems by combining primitive objects using the following boolean operators: union, intersection and difference. Fig. 4.7 shows how CSG can be used to construct a simple object, a mug, by combining primitive solids using these operators. CSG systems usually are restricted to working with *regular* solids; a set of points, X, is said to be regular if it is equal to the closure of its interior, that is

$$X = ki(X)$$

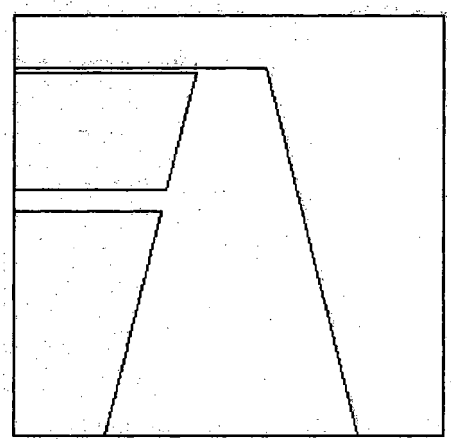
where k and i denote the closure and interior, respectively. Because a solid produced by the combination of regular solids using the *set-theoretic* boolean operations is not necessarily regular, CSG systems use *regularized* boolean operators when combining objects to guarantee that the result of the combination will be regular. Fig 4.8 shows how a nonregular object can result from the set-theoretic intersection of two regular objects; it also shows the object produced by the regularized intersection of the two objects. The set-theoretic intersection of the two faces in panel (a) of Fig. 4.8 is shown in panel (b); note that the result of the combination is not regular (because of the "dangling" edge). Panel (c) shows the valid face produced by taking the regularized intersection of the two faces in panel (a). The set-theoretic union and difference operators have similar problems. The regularized operators, union ($\cup^*$), intersection ($\cap^*$) and difference ($-^*$), of two sets, X and Y, are defined as

$$X \cup^* Y = ki(X \cup Y)$$

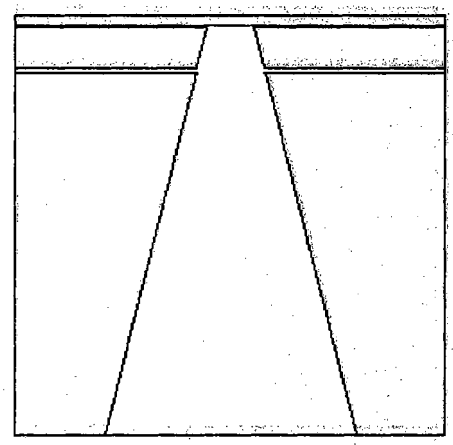
$$X \cap^* Y = ki(X \cap Y)$$

$$X -^* Y = ki(X - Y)$$

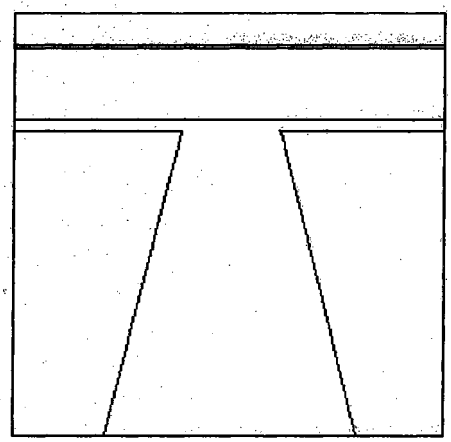
Most of the concepts used in CSG modeling systems were originally developed for the PADL solid modeling system [VoeReq77], [HarMar85].



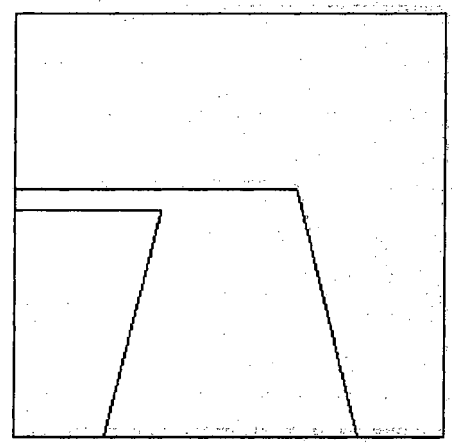
position 1



position 2



position 3



position 4

FIGURE 4.6 This figure shows some typical expected scenes generated for a mobile robot with downward pointing cameras. The scenes depicted in this figure were generated with the map shown in Fig. 4.2.

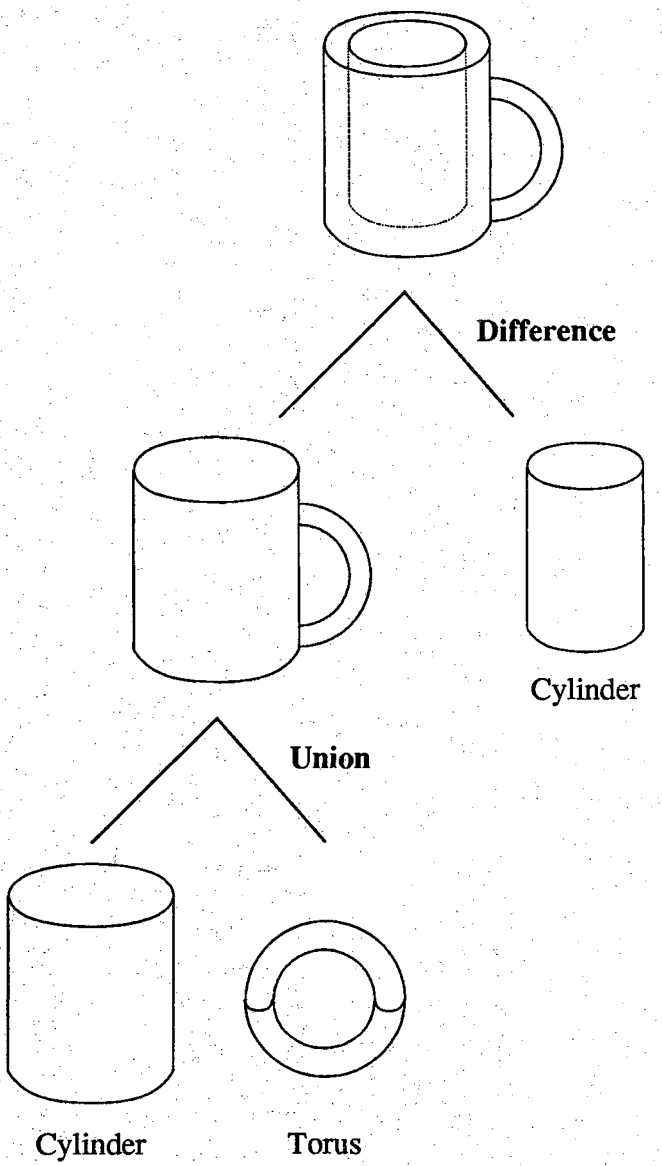


FIGURE 4.7 This figure demonstrates how objects are defined in CSG systems by the boolean combination of successively simpler objects. The coffee mug in this figure is defined in terms of two cylindrical primitives and one toroidal primitive.

Boundary-representation modeling is another common solid-modeling technique. In this scheme, objects are represented in terms of their boundary surfaces. In many B-rep systems, polyhedrons are used to approximate the boundary of the objects; thus, any curved surfaces, such as cylindrical or spherical surfaces, are only approximately represented. PSEIKI uses the TWIN B-rep solid modeling package [Mas87] to generate expected scene information in an industrial domain. TWIN was developed at the Computer Aided Design and Graphics Laboratory (CADLAB) at Purdue University's Engineering Research Center. TWIN is a library of

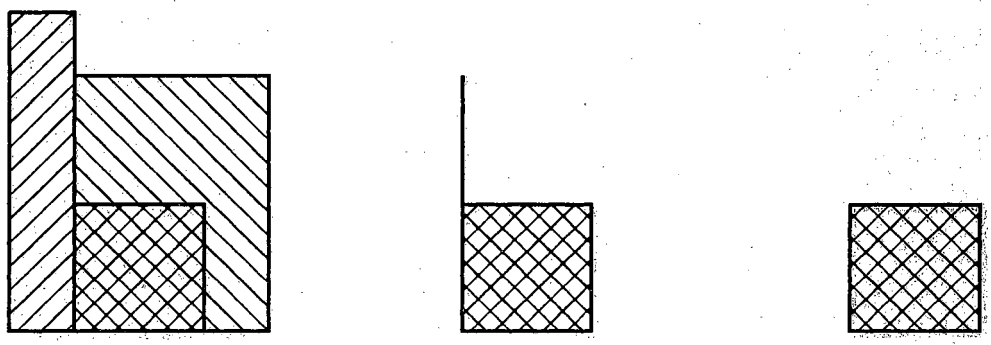


FIGURE 4.8 This figure shows a shortcoming of the set-theoretic intersection operation in two dimensions. The face in panel (b) is the set-theoretic intersection of the two faces in panel (a); it is not a valid face (because of the dangling edge). The face in panel (c), a valid face, is the regularized intersection of the two faces.

subroutines in the C language that contains routines to generate the primitive objects included in most CSG systems; the set of primitives that TWIN can generate includes parallelepipeds, wedges, cylinders, cones, toruses, spheres, fillets, elliptical cones, and ellipsoids. The library also contains routines to perform regularized boolean operations on solid objects. Because the TWIN library contains routines to generate the primitives used in CSG systems and routines to perform the operations used by CSG systems, the same process used to generate solid objects in CSG systems can be used to generate objects with TWIN. That is, solid objects can be defined by regularized boolean combinations of primitive objects.

A two step procedure is used to convert the TWIN models into a form usable by PSEIKI. First, the Watkins scan-line rendering algorithm [Wat70], is used to generate an image of the expected scene. The grey value of every pixel in the rendered image is set to the id number of the model surface visible at that location in the image; thus, regions in the image with the same grey level all belong to the same surface in the TWIN model. After the model is converted into an image, the region-based preprocessor described in chapter 3 is then used to extract the image's labeled regions and output the scene description on the vertex, edge and face levels. The threshold values required by the segmenter are set to zero so that each region detected by the segmenter will correspond to a single model surface. The information on the object and scene levels is generated by assuming that there is only a single object in the expected scene. Thus, all of the faces detected in the image, with the exception of the background face (which has id number zero), are grouped into a single object. This object is then set to be the only object in the scene. If there is more than one object in the image, then the output file must be hand edited to correct the object level and scene level information. Fig. 4.9 shows a graphic output of the system for an industrial object, a piston connecting rod; this figure shows three orthogonal views and one oblique view of the object. Fig 4.10 illustrates the process used to

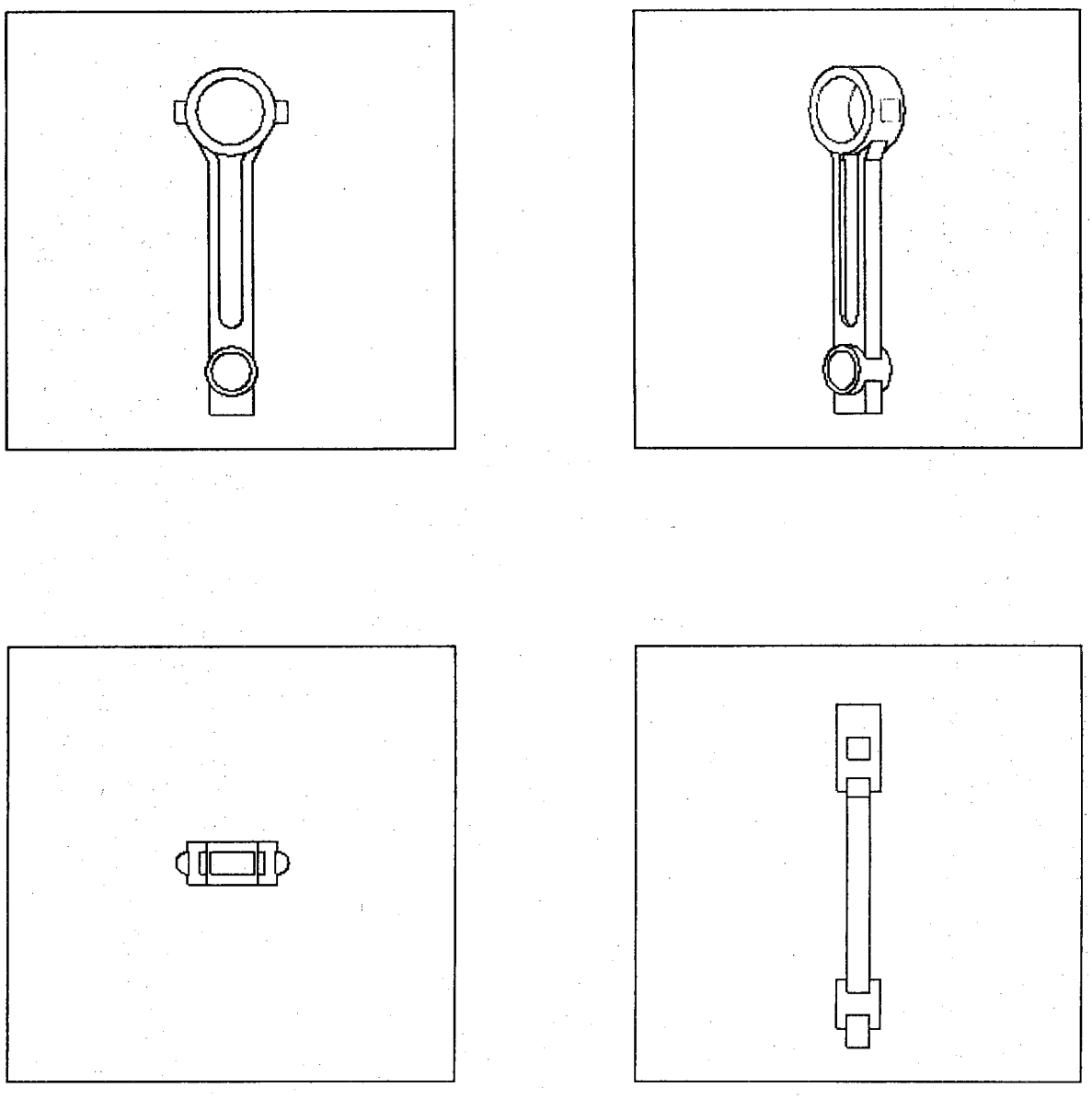
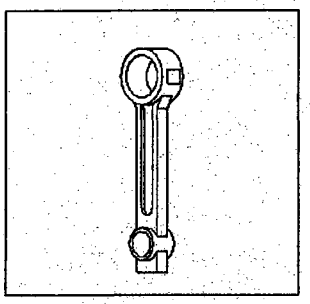


FIGURE 4.9 This figure shows some typical expected scenes generated in an industrial environment. It shows a piston rod from three orthogonal views and one perspective view.

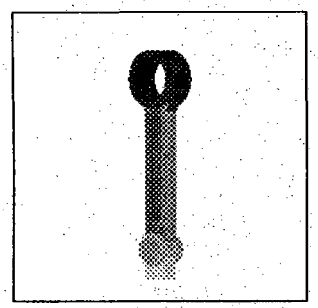
generate the data file for the oblique view of the connecting rod shown in Fig. 4.9. The image at the top of the figure represents the TWIN solid model. The image at the middle of the figure shows the rendered image with uniquely labeled surfaces. A small portion of the symbolic output is shown at the bottom of the figure. In reality, this data file contains the definition for about 200 elements.

The current method of generating PSEIKI's expected scene information has a number of obvious flaws. The main deficiency of the technique is that all 3D information is lost when the model is rendered. This deficiency has not been a problem to date because of the simple scenes currently being used to test PSEIKI; this loss of information is expected to become more limiting as PSEIKI is applied to more complex scenes in the future. Another deficiency of the technique is the assumption that there is only one object visible in the expected scene. Although it is usually not difficult to correct this information by hand if there is more than a single object in the scene, it would be convenient if the system was able to correctly generate PSEIKI's input data at all levels of abstraction. Future versions of the expected scene generator will avoid these problems by converting the expected scene information directly into a form that PSEIKI can use without the intermediate rendering step. These future versions of the expected scene generator will be able to easily convert curved borders of nonplanar surfaces into piecewise-linear segments for PSEIKI's input because the curved borders are already represented as piecewise-linear facet boundaries in the TWIN models.



TWIN solid model

Watkins rendering algorithm



Rendered image

Region-based Segmentation

```
+ scene id 2194 children 2193
+ object id 2193 children 2174 2175 2176 2177 2178 2179 2180 2181 2182 2183 ...
+  face id 2174 children 2149 2161 2162 2163 2164 size  684 value 2
+  face id 2175 children 2103 2145 2146 2147 2148 2149 size  924 value 255
+  face id 2176 children 2106 2107 2108 2109 2142 2143 2145 2146 2155 2156 ...
+  face id 2177 children 2084 2085 2086 2087 2088 2089 2090 2091 size  1098 value 5
⋮
```

FIGURE 4.10 This figure shows the processing performed to generate the symbolic expected scene data for industrial objects. The top image represents the TWIN solid model information; the middle panel shows the rendered model image with every surface uniquely labeled. The lower part of the image shows a small portion of the symbolic output which would be presented to PSEIKI.

CHAPTER 5

AN EVIDENCE ACCUMULATION SCHEME FOR BLACKBOARD REASONING

The use of inexact reasoning in computer vision systems is certainly not new; however, most of the previous schemes for evidence accumulation have been based only loosely on formal uncertainty calculi [HanRis78], [MckHar85]. The main reason that these systems employed ad-hoc schemes is the overwhelming amount of data in an image; the systems needed a fairly simplistic evidence accumulation scheme to avoid becoming bogged down in confidence value computations. In contrast, the evidence accumulation scheme used in PSEIKI is based on the Dempster-Shafer (D-S) theory of evidence,* whose normally exponential computational complexity is controlled by a number of mechanisms to be discussed in this chapter. For example, one of the mechanisms consists of accumulating evidence over binary sets of hypotheses, meaning the evidence either supports that a data element from the image should be given a particular label from the model or denies this supposition. Pooling of evidence in this fashion leads to a particularly efficient implementation of the Dempster's rule. Performance also is improved by exploiting the hierarchical nature of the blackboard system. By performing a small number of computations on upper levels of the hierarchy, many computations on lower levels can be avoided. The hierarchical nature of the blackboard also is used to constrain the matching process for elements on lower levels of the hierarchy; elements on lower levels of the hierarchy are allowed to match only if their parents are matched.

In the next section of this chapter, we will introduce Dempster's rule of combination and point to its exponential time complexity. Next, it will be shown how, in past systems, the computational efficiency of Dempster's rule was improved by making assumptions that the focus of incoming evidence is limited to a small number of subsets of Θ. Once these assumptions are made, a computationally efficient form of Dempster's rule can be derived. The evidence accumulation scheme employed by PSEIKI will be introduced in this context; the new accumulation scheme will first be introduced in general terms. Next, it will be shown that the accumulation scheme can be embedded into a hierarchy if the reasoning task has the appropriate structure. It also will be shown that the hierarchical structure allows the computational complexity of the scheme to be improved by limiting the size of the elements' FODs and by limiting the number of evidence sources that are allowed to provide updating evidence. Furthermore, a method for passing belief values up and down the hierarchy will be introduced. After the general scheme has been fully developed, its use by PSEIKI's labeler KS will then be presented as an application. Finally, to show the enerality of our evidence accumulation scheme, we will point out how it could be applied to the speech recognition domain.

* It is assumed that the reader has a working knowledge of the Dempster-Shafer theory of evidence and its associated terminology. For those not familiar with the theory, see [Sha76]; a brief review is also presented in appendix A.

5.1. Computationally Feasible Methods For Evidence Accumulation Based on the Dempster-Shafer Theory

The Dempster-Shafer theory of evidence is gaining wider acceptance as an uncertainty calculus. However in the general case the formula used to accumulate evidence in this theory, Dempster's sum, takes exponential time (in the size of the FOD) to combine evidence from two independent sources. This is shown easily by observing the formula for Dempster's sum as shown in equation 5.1.

$$m(X) = m_1 \oplus m_2 = K \sum_{\substack{X_1 \, X_2 \\ X_1 \cap X_2 = X}} \sum m_1(X_1) m_2(X_2) \tag{5.1}$$

where

$$K^{-1} = 1 - \sum_{\substack{X_1 \, X_2 \\ X_1 \cap X_2 = \varnothing}} \sum m_1(X_1) m_2(X_2)$$

The main reason for the exponential complexity is the requirement that the probability mass for all $2^{|\Theta|}$ elements of the power set of Θ be evaluated when combining evidence from independent sources. If N bpa's are combined to form a data-element's belief function, then the total number of operations will be on the order of $N \times 2^{|\Theta|}$ (this will be denoted as $O(N \times 2^{|\Theta|})$).

Barnett [Bar81] was one of the first to show that Dempster's rule could be implemented in better than exponential time if the focus for all evidence is restricted to a limited number of elements of 2^{Θ}. Barnett was able to implement Dempster's rule in linear time by assuming that all evidence either confirms or denies members of the FOD. Although this assumption places a fairly large restriction on the general D-S theory, many systems naturally provide evidence in this form and are not hindered by the assumption. It can be shown that, in the general case, $O(N \times |\Theta|)$ operations are required to combine N bpa's when using Barnett's equations.

Gordon and Shortliffe also were able to improve the computational complexity of the D-S theory by making an assumption about the type of evidence allowed to update beliefs [GorSho85]. They formed what they termed a hierarchical hypothesis space, a hierarchical partition of an element's FOD, and assumed that all evidence either would confirm or deny elements in the partition. An example of a hierarchical hypothesis space that could be used in a computer vision system is shown in Fig. 5.1; it shows the partition that could be used by a target identification system to classify tactical objects detected in a sequence of image frames. The identification system could use the partition shown in this figure if it detected an object that moved from frame to frame. If the system detected a moving oject, then it would be able to use the hierarchy to provide evidence asserting that the object was a vehicle without needing to specify which type of vehicle. A system using the formulas derived by Barnett would not be able to provide evidence for the generic class of vehicles directly, because evidence is limited

to focusing on the individual members of the FOD in that scheme.

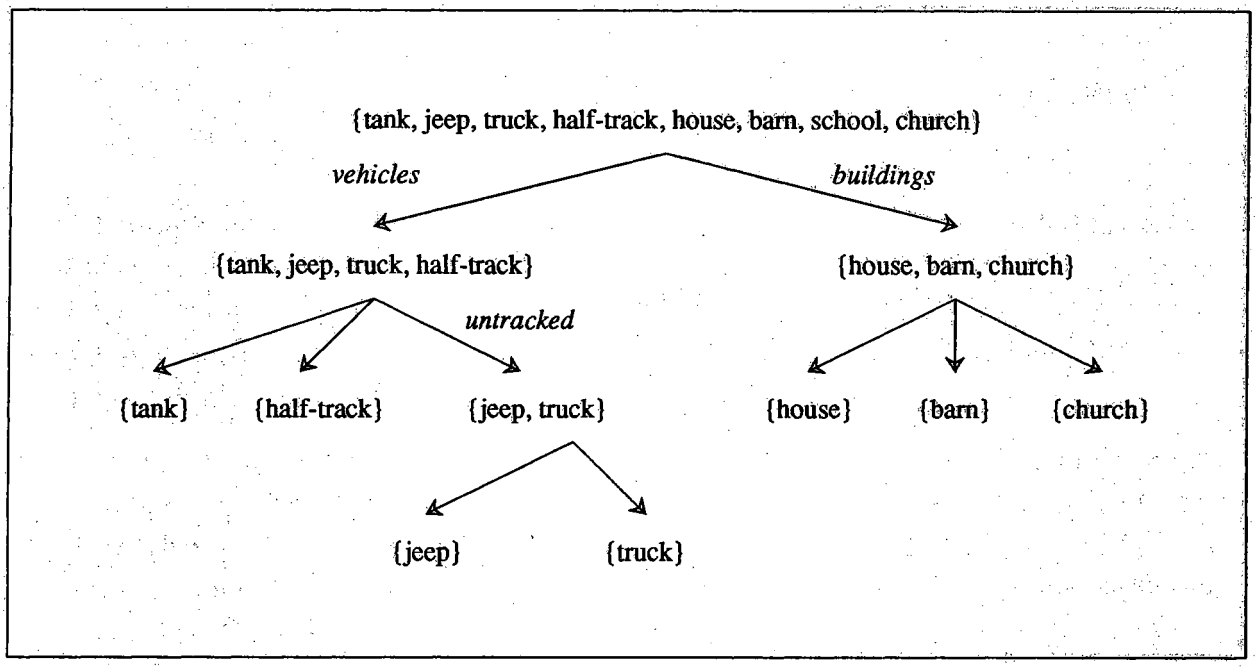


FIGURE 5.1 This figure shows a hierarchical partition of a hypothesis space that could be used to classify objects detected in a tactical image.

In order to provide a computational gain, Gordon and Shortliffe were forced to approximate Dempster's sum; the resulting approximation had a number of drawbacks. When presented with highly contradictory evidence, the approximation produced poor results. The approximation also prevented the computation of belief values for negations of elements in the hierarchy; thus, plausibilities for elements in the hierarchy could not be computed.

Shafer and Logan were able to formalize the problem of using Dempster's rule to combine evidence focused on elements of a hierarchical partition [ShaLog87]. By applying variations of Barnett's formulas to elements in a hierarchical partition of an element's FOD, they were able to compute Dempster's sum for elements in the partition without any approximations; thus, the results that their formulas provide are always valid. Their formulas also are slightly more general than those used by Gordon and Shortliffe in that they can compute both belief values and plausibilities for elements in the hierarchy.

Binary frames of discernment (BFODs) involve the most drastic restriction to the D-S theory, but they provide the greatest computational gain. As presented in [SafGot87], a BFOD is a FOD with $|\Theta| = 2$. Equivalently, any FOD with all of its probability mass constrained to two disjoint elements of 2^{Θ} and $\{\Theta\}$ itself can be thought of as a BFOD. Obviously, since the size of the FOD is constrained to be 2, the time needed to combine two bpa's is constant. Thus the time needed to combine N bpa's is O(N).

Although the above variations of Dempster's rule greatly improve its computational efficiency, none of them are applicable to the problem of accumulating evidence in PSEIKI. Binary FODs are too restrictive to be used in a general matching procedure; their requirement that all probability mass be constrained to three subsets of Θ severly limits their applicability. Barnett's scheme, while remaining general enough for use in PSEIKI, is still too inefficient to handle the overwhelming amount of data in an image. Finally, the use of hierarchical hypothesis spaces is not possible because the hierarchy used in PSEIKI is not strict (e.g. an edge can be the children of two faces -- the faces it separates). For this reason, a new procedure to accumulate evidence was developed by incorporating the concept of an element's label into the reasoning process.

5.2. A Computationally Efficient Evidence Accumulation Scheme using Labels

As has already been mentioned, Barnett was able to implement Dempster's sum in linear time (with respect to the size of the FOD) by assuming that all evidence either confirms or denies members of the FOD. Although this is a great improvement over the complexity of the original formulation of Dempster's rule, in the general case it still takes $O(N\times|\Theta|)$ operations combine the evidence contained in N bpa's. We will show that it is possible to further reduce the computational complexity of Dempsters rule by splitting the accumulation process into two phases: initialization and updating. In the accumulation scheme introduced here, a single evidence source is used to define the bpa during the initialization phase. This source is used to provide confirmatory and disconfirmatory evidence focused on all members of the FOD, much as in Barnett's scheme. Once the belief function has been established, the updating phase commences and the focus of all new evidence is restricted to focus on only particular elements in the FOD. We will show that, by using this two phase accumulation scheme, it is possible to combine evidence from N sources in $O(N+|\Theta|)$ operations.

In this accumulation scheme, assume that the identity of an element, E_i, is in question and that its identity can be any one of M possibilities, $\theta_1, \theta_2, \cdots, \theta_M$. Therefore, the FOD for E_i is

$$\Theta = (\theta_1, \theta_2, \cdots, \theta_M)$$

Furthermore, assume that there are N evidence sources, $S_1, S_2, \cdots, S_N$, that can provide information about the element's identity. The sources are assumed to provide bpa's with evidence that is focused entirely on members of Θ or their compliment. This is exactly the same assumption that Barnett uses to reduce the computational complexity of Dempster's rule to linear time. However, we also assume that it is also possible to force the evidence sources to provide bpa's with evidence focused entirely on a single member of Θ and that element's compliment*. It is this ability to restrict the focus of the evidence produced by the sources that

* One way forcing the bpa's to have this form is to incorporate all the probability mass from unwanted subsets of Θ to Θ itself.

enables the computational complexity of the accumulation scheme to be reduced.

During the initialization phase of the accumulation process, a single evidence source is used to provide evidence about E_i's label. This source is used to define the initial bpa by providing confirmatory and disconfirmatory evidence focused on all (singleton) members of the FOD, as in Barnett's scheme. After the initial bpa for the element's identity is computed, the **label** of the element is determined. An element's label is defined to be the member of Θ with the largest belief. (If two or more elements of the FOD yield the same belief, one is arbitrarily selected). Thus, in some sense, the element's label can be considered to be the current best hypothesis for the element's identity. As an example of the ease with which an element's label can be found, consider the process of determining the label for element E_i. Remember that the FOD for E_i consists of M elements

$$\Theta = \{\theta_1, \theta_2, \cdots, \theta_M\}$$

To determine the E_i's label, the element of Θ with maximum belief must be found. However, since only singletons are being considered as labels, finding the element of Θ with greatest belief is equivalent to finding the element of Θ with the largest probability mass (the belief of a singleton is equal to its probability mass). Thus, only the following elements of E_i's bpa need be considered

$$m_{E_i}(\{\theta_\alpha\}) \quad \text{for} \quad \alpha = 1, ..., M$$

Let $\theta_{\alpha_{max}}$ be the element of Θ for which the bpa takes a maximum value. That is,

$$m_{E_i}(\{\theta_{\alpha_{max}}\}) \geq m_{E_i}(\{\theta_\alpha\}) \quad \text{for} \quad \alpha = 1, ..., M$$

The label of element E_i is defined to be $\theta_{\alpha_{max}}$.

Once the initialization phase of the accumulation scheme is complete and the initial bpa for E_i's identity has been computed and the label determined, the belief updating phase begins. In this phase, all evidence is restricted to focus on the label-element and its compliment. That is, the only elements in an updating bpa that are allowed to have non-zero probability masses are $\{\theta_{\alpha_{max}}\}$, $\{\neg\theta_{\alpha_{max}}\}$ and $\{\Theta\}$. Thus at any time, all new evidence provided by the evidence sources is focused on either trying to prove or trying to disprove that an element's label is correct (i.e. that the element's identity has been correctly determined). If the accumulated disconfirmatory evidence about an element's label is enough to force the belief in the label element to fall below the belief in another member of Θ, then the label will be changed to the element with greater belief. The evidence sources then are allowed to provide confirmatory or disconfirmatory evidence about the new label.

When incorporating new evidence for an element's label, the computational load can be eased by making use of the associative nature of Dempster's rule. If an element's bpa is updated incrementally with every piece of new evidence, as is done in Barnett's scheme, then the new bpa will be computed as

$$m_{new} = (((m_{old} \oplus m_{update_{2\rightarrow i}}) \oplus m_{update_{3\rightarrow i}}) \oplus \cdots \oplus m_{update_{N\rightarrow i}})$$

Where $\oplus$ denotes Dempster's rule of combination and $m_{update_{j\rightarrow i}}$ is the updating bpa for the new evidence that source S_j is providing for element E_i. Since Dempster's rule of combination is invariant with respect to the order of combination, the new bpa can be expressed as

$$m_{new} = m_{old} \oplus m_{update}$$

where

$$m_{update} = ((m_{update_{2\rightarrow i}} \oplus m_{update_{3\rightarrow i}}) \oplus \cdots \oplus m_{update_{N\rightarrow i}})$$

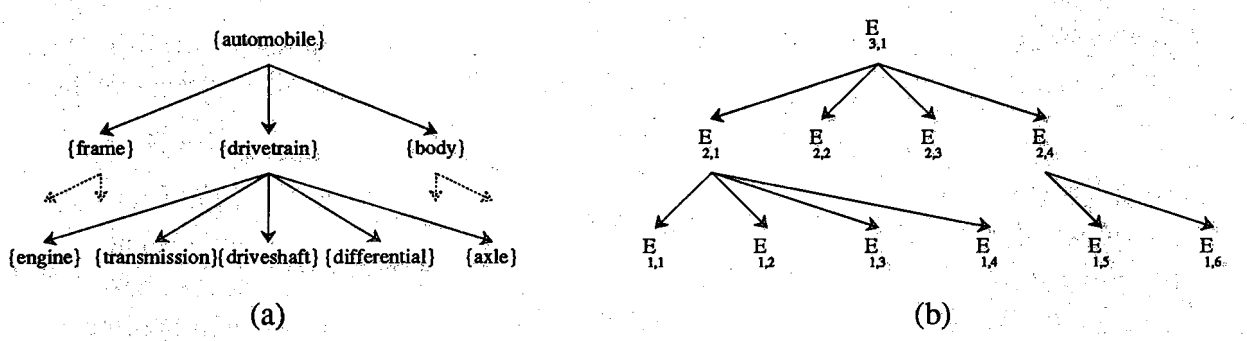
Now the fact that the updating bpa's use binary frames of discernment can be exploited (if element E_i is labeled as E_A then the only elements of E_i's bpa with nonzero probability mass correspond to the subsets $\{E_A\}$, $\{\neg E_A\}$ and $\{\Theta\}$ itself). Because BFODs can be updated in constant time, the time needed for N evidence sources to form an element's updating bpa is O(N). Furthermore, Barnett's formulas can be used to incorporate the updating bpa into the initial bpa in $O(|\Theta|)$ time. Thus, the total amount of time for N evidence sources to update an element's belief in its label is $O(N+|\Theta|)$. Of course, if an element's label changes during the accumulation process then further computation is necessary because the evidence sources must provide evidence to update the element's belief in the new label.

It should be mentioned that if the updating belief for a number of elements is generated by noting the degree to which their labels are mutually compatible, then the updating evidence contained in their updating bpa's should not be incorporated into their belief functions until all of the updating bpa's are formed. If the incorporation of the updating bpa's is not delayed in this manner, then it is possible for the updating bpa for an element to be influenced by its belief in its own label. An element could provide updating evidence to itself if it was used to generate updating evidence in another element's label which in turn was then used to provide updating evidence about the first element's label. Delaying the incorporation of updating evidence into elements' belief functions until all updating evidence has been generated prevents this from occurring.

5.3. Hierarchical Evidence Accumulation in PSEIKI

If the task of a system is to determine the identity of a number of elements which are arranged into a *part-of* hierarchy, then the evidence accumulation scheme introduced in the previous section can be embedded into the hierarchy to provide further computational gain. A part-of hierarchy is shown in panel (b) of Fig. 5.2; in this figure, as in most part-of hierarchies, elements on the higher levels of the hierarchy are defined by groups of elements on lower levels. For example, in this figure elements $E_{1,1}$ through $E_{1,4}$ are grouped to form element $E_{2,1}$ ($E_{i,j}$ denotes the j^{th} element on the i^{th} level of the hierarchy). Part-of hierarchies are a natural

way to represent many types of objects. For example, this report contains a number of chapters each of which, in turn, contain a number of sections. As we progress down this structure, we find that the sections can be broken down into paragraphs, sentences, clauses, words and letters. An automobile also can be represented hierarchically using a part-of hierarchy. Panel (a) of Fig. 5.2 is a simple example of how an auto can be broken down into its major assemblies (the frame, the body and the powertrain) and how each of these assemblies can be broken down into its main components. Of course, a part-of hierarchy that could be used to represent a real auto would be much more complex. Note that these hierarchies do not need to be strict; that is, an element can have more than a single parent if it is in more than one group.



(a) (b)

FIGURE 5.2 (a) Hierarchical description of an automobile. (b) demonstrates how a number of unidentified elements can be grouped into a part-of hierarchy.

The structure of a part-of hierarchy can be used to aid in the determination of the identity of its elements. For example, in many cases, the label for an element will dictate the possible labels that its children can assume. For example, in Fig. 5.2, if elements $E_{1,1}$ through $E_{1,4}$ are grouped to form element $E_{2,1}$ and if $E_{2,1}$ is thought to be the drivetrain of an auto, then the possible labels for elements $E_{1,1}$ through $E_{1,4}$ would be

$$\Theta = \{\text{engine, transmission, driveshaft, differential, axle}\}$$

Thus panel (a) of Fig. 5.2 can be thought of as a hierarchical arrangement of the possible labels that the elements of panel (b) can assume (i.e. their frames of discernment). If the hierarchy was not used to restrict certain possible labels from being included in an element's FOD, then the FOD might include all possible labels on the same level of the hierarchy. As it stands, the FOD for an element is determined by its parent's label-element and the children of its parent's label-element. Specifically, an element's FOD is defined to be the children of its parent's label-element.

Because an element's FOD is determined by its parent's label, the FOD for the element and all of its descendents must change if the parent's label changes -- a computationally intensive operation. Thus it is advantageous to incorporate new evidence on upper levels of the hierarchy before incorporating evidence on lower levels of the hierarchy. Because calculations on upper and lower levels of the hierarchy can be thought to correspond to checking global and

local consistencies respectively, generating updating evidence for elements on the upper levels of the hierarchy before generating updating evidence for elements on lower levels corresponds to performing global consistency checks before local ones.

To further curtail the number of uncertainty calculations, elements are only used to generate updating evidence for their siblings. For example, only elements thought to be part of the auto's drivetrain would be used to generate updating evidence for other elements in the drivetrain. If the data were not arranged hierarchically, every element would be needed to generate updating evidence for every other element. An element can be used to generate evidence for another element that is not a sibling by propagating the first element's confidence value up through the hierarchy until a common ancestor is reached and then back down to the second element.

5.3.1. Evidence Propagation Between Levels in the Hierarchy

Evidence from an element's siblings is not the only source of knowledge used to update its belief function. A mechanism also is provided for passing belief values between different levels of the hierarchy. This is done to satisfy the intuitive argument that says any evidence confirming an element's label also should provide evidence that its parent's label is correct. Disconfirming evidence also is required to be passed down to the lower levels of the hierarchy Furthermore, it is intuitively appealing to pass both confirmatory and disconfirmatory information up the hierarchy if all updating evidence for an element is generated by measuring its consistency with its siblings.

The updating bpa, m_{update}, is used when passing evidence up the hierarchy. To do this, m_{update} is combined not only with the bpa for the element in question, but also with that for the element's parent. Combining the updating bpa with an element's parent makes intuitive sense because all new evidence generated on a level comes from the (in)compatibility between elements on that level. If the children of an element have consistent (compatible) labels, then these child-elements should provide evidence that the label given to the parent-element is correct. Likewise children with inconsistent labels provide evidence that their parent's label is incorrect. Thus, by passing the updating bpa's to each parent-element, new evidence is provided for those elements based on the consistency or the inconsistency of their descendents.

Evidence from an element cannot be applied directly to its parent because the FODs of an element and its parent are composed of different types of data elements. However, it will be shown, with the help of the above example, that it is possible to build a FOD that can be used to update the belief functions of elements on a higher level of the hierarchy. Assume that the data is as shown in panel (b) of Fig. 5.2 and that element $E_{1,1}$ is a child of element $E_{2,1}$. Furthermore, assume that $E_{1,1}$ is labeled as the transmission and $E_{2,1}$ is labeled as the drivetrain. Because the confirmatory evidence for $E_{1,1}$'s label derived from its siblings arises

from the consistency of the label with its sibling's labels, it may be considered as a weighted vote of confidence that $E_{2,1}$'s label is correct. Likewise, because the disconfirmatory evidence for $E_{1,1}$'s label derived from its siblings arises from the inconsistency of the label with its sibling's labels, it may be considered as a (weighted) vote of no confidence in $E_{2,1}$'s label. Thus, $m_{\underset{E_{1,1}}{update}}(\Theta)$ can be considered to be the amount of ignorance in $E_{2,1}$'s label. Using this rationale, an updating bpa for $E_{2,1}$ with the following non-zero probability masses may be defined as

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\text{drivetrain}\}) = m_{\underset{E_{1,1}}{update}}(\{\text{transmission}\})$$

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\{\neg\text{drivetrain}\}) = m_{\underset{E_{1,1}}{update}}(\{\neg\text{transmission}\})$$

$$m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}(\Theta_{E_{2,1}}) = m_{\underset{E_{1,1}}{update}}(\Theta_{E_{1,1}})$$

Now it will be shown that $m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}$ is a bpa for $E_{2,1}$. As described in the last section, the FOD for an element's updating bpa is binary in nature. Thus the only non-zero elements of the updating bpa for E_1 are $m_{\underset{E_{1,1}}{update}}(\{\text{transmission}\})$, $m_{\underset{E_{1,1}}{update}}(\{\neg\text{transmission}\})$, $m_{\underset{E_{1,1}}{update}}(\{\Theta\})$ and they sum to 1. Because $m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}}$ has unity total mass and its null hypothesis has zero mass, it is a bpa by definition. The total accumulated new belief for $E_{2,1}$ from its children $E_{1,1}$, ..., $E_{1,4}$ now can be expressed as

$$m_{\underset{E_{2,1}}{update}} = (m_{\underset{E_{1,1}\rightarrow E_{2,1}}{update}} \oplus \ldots \oplus m_{\underset{E_{1,4}\rightarrow E_{2,1}}{update}})$$

Information is passed down the hierarchy only if it is disconfirmatory. This downward propagation of information takes the form of the reassignment of frames of discernment caused by the ancestor of an element having its label changed. In the previous example, this could happen if the hypothesized identity for $E_{2,1}$ is changed to be the frame of the auto. Using information from the model panel, the FOD for E_1 would be reassigned to

Θ = {carraige, front suspension, rear suspension}

It should be mentioned that there are two cases that require special consideration. First, a data-element may have no siblings; in this case, since the element's consistency can not be checked with its siblings, the only updating evidence that will be received about its label will be generated by checking the consistency of its children's labels. The other special case occurs when an element's label-element is an only child; in this case, there is only one member of the element's FOD. Therefore, the element's label can not be changed no matter how small the belief in this label becomes. Note that since the element has only one element in its FOD ($|\Theta|=1$), its bpa is a simple support function.

5.4. Use of the Hierarchical Evidence Accumulation Scheme in PSEIKI

The evidence accumulation scheme introduced here was originally developed to aid in the matching of data-elements with model-elements by PSEIKI's labeler KS. In this application, the labeler KS uses the scheme to determine the identities of the elements on the data panel of the blackboard. Their possible identities are the elements on model panel. To illustrate how the scheme used used by PSEIKI, consider the example in Fig. 5.3.

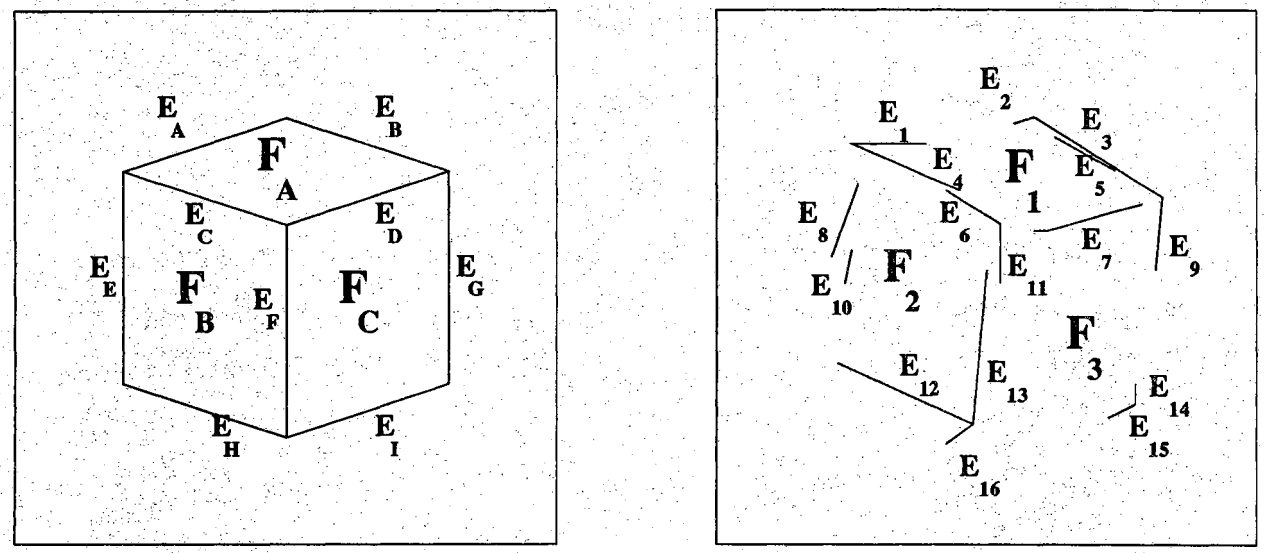


FIGURE 5.3 The left panel of this figure shows a simple example of model-elements derived from a graphics source; the right panel shows image-elements from 2D vision data. The figure is used to aid the textual explanation of how PSEIKI's labeler KS uses the evidence accumulation scheme introduced here. Note that the elements in this figure could represent only a small fraction of the data-elements on the blackboard panels.

This figure shows the edge-level and face-level of the data on the blackboard. Model-data is shown in the left panel; in this frame edges E_A through E_D are grouped into face F_A. The right panel shows image-data; here edges E_1 through E_8 are grouped into face F_1*.

As was mentioned in the previous section, the hierarchical nature of the matching task is used to increase the efficiency of the matching process by restricting the model-elements allowed to be members of an image-element's FOD. For example, in Fig. 5.4, if F_1 is matched with F_A, then the frame of discerment for edges E_1 through E_8 would be

$$\Theta = \{E_A, E_B, E_C, E_D\}$$

These model elements are allowed be members of the FODs for edges E_1 - E_8 because they

*Note that in the following discussion the elements generated by the graphics source have capital letters as subscripts while elements derived from 2D vision data have numeric subscripts.

are the children of their parent's model element, face F_A.

The hierarchical nature of the task is exploited further by checking the consistency of an element only with its siblings. In the previous example, the belief E_1's label would be updated only with evidence generated by noting its consistency with edges E_2 - E_8. These edges would be used to provide the updating evidence because they are grouped into face F_1 along with edge E_1. If the image-elements were not grouped hierarchically, then every edge would be needed to generate updating evidence in E_1's label. The method used to generate updating evidence for edges and faces based in their consistency with their siblings is discussed in chapter 6.

PSEIKI's labeler KS also propagates updating bpa's up the hierarchy in the previously discussed manner. For example, if edge E_1, one of face F_1's children, has label E_A and face F_1 has label F_A, then the following updating bpa for F_1 can be created from the updating bpa for E_1

$$m^{update}_{E_1 \rightarrow F_1}(\{F_A\}) = m^{update}_{E_1}(\{E_A\})$$

$$m^{update}_{E_1 \rightarrow F_1}(\{\neg F_A\}) = m^{update}_{E_1}(\{\neg E_A\})$$

$$m^{update}_{E_1 \rightarrow F_1}(\Theta_{F_1}) = m^{update}_{E_1}(\Theta_{E_1})$$

The bpa's are propagated upwards for the reasons discussed earlier. Compatibly labeled siblings should provide confirmatory evidence about their parent's label; conversely, incompatibly labeled siblings should provide disconfirmatory evidence about their parent's label. As in the general scheme, changing the bpa for an element on an upper level of the hierarchy will force all of its descendents to change their FODs. The FODs are changed to satisfy the heuristic which states, for example, that the constituent edges of a mislabeled face also are most likely mislabeled.

5.5. Another Application of the Hierarchical Evidence Accumulation Scheme

It is also possible to use the hierarchical evidence accumulation scheme developed here in domains suitable for blackboard processing other then computer vision. The scheme is applicable to these domains because of their hierarchical nature. For example, the evidence accumulation scheme could be used in the domain for which the Hearsay-II [ErmHay80] blackboard system was developed: speech understanding. We will examine how the evidence accumulation scheme could be used by a speech understanding system based on Hearsay-II.

Speech is represented hierarchically in the Hearsay-II system on the following 6 levels: *phrases*, *word-sequences*, *words*, *syllables*, *segments* and *parameters*. The lowest-level of the representation, the parameter level, breaks the speech waveform into five classes: silence,

sonorant peak, sonorant nonpeak, fricative and flap. The next higher level, the segment level, is used to label the elements on the parameter level with phoneme-like labels. These labels are generated using statistical pattern recognition techniques and can assume 98 different values. Hearsay-II forms the elements on the higher levels of the hierarchy (the syllable, word, word-sequence and phrase levels) by grouping compatible elements from the lower levels.

To apply the accumulation scheme to Hearsay-II's task, the statistically-based classifier could still be used to generate phoneme-like labels for the parameter elements. However, initial belief values for the segments' labels could be generated from the probabilities produced by the segment classifier. Updating evidence for the elements' labels could then based on the compatibility between the elements and their siblings, as is done in PSEIKI. For example, on the word level of the blackboard, if an adjective is followed by a noun then the two should lend support to each other.

Updating evidence could be passed up the hierarchy as is done in PSEIKI (for example, evidence that a word is correct would also be evidence that its parent phrase is correct). Likewise, changing the label of an element on an upper level of the blackboard would cause all of its descendents to change their FODs.

5.6. Future Work

In this chapter, a new hierarchical evidence accumulation scheme based on a restricted form of Dempster's rule has been developed and informally has been shown to be computationally efficient. This efficiency has been shown to stem directly from a restriction on the focus of updating evidence; however, research needs to be performed on how restrictions placed on the updating evidence affect an element's belief function. Future work will investigate how an element's belief function changes when the various schemes discussed in this chapter are used for evidence accumulation. This investigation will use techniques developed in the past to compare competing models for inexact reasoning [MitHar87]. Another topic that may warrant investigation is the performance of the combination scheme when the evidence provided by the sources is not independent. Some previous work addressing this topic can be found in [DubPra85], [DubPra86], [HunJay87], [Kyb87], [Sme76], and [Yen86].

CHAPTER 6

GEOMETRIC COMPUTATIONS FOR INITIAL AND UPDATING BELIEF FUNCTIONS

Chapter 5 showed how evidence is used to generate and update belief in a data-element's label; however, no mention was made of how that evidence is generated. This chapter will address the process of generating evidence to choose initial labels for elements and to update the confidence values for those labels. In PSEIKI, evidence about an element's label is generated by measuring how well the element meets geometric constraints between itself and other elements. These constraints take two general forms. Initially when matches are being formed, the constraints measure the similarity between an image element and model elements. After the initial matches are found and a label for the element has been determined, the constraints are used to measure how consistent the element's label is with the labels of its siblings in the hierarchy.

There are many techniques available that PSEIKI can use to determine if elements are meeting geometric constraints. Besl describes some general techniques to match image data and model data at various levels of abstraction (points, curves, surfaces and volumes) using geometric constraints [Bes88]. Crowley and Ramparany take a different approach to the process of generating evidence based on geometric constraints; they model sensor readings as samples from a multivariate Gaussian distribution and use this assumption to calculate a "distance" from a feature measurement to its mean value [CroRam87]. They then estimate the belief in an entity based on the distance measured. No matter what method is used to measure the degree to which the elements are meeting the geometric constraints, the constraint measurements must be converted into belief functions. The method used in PSEIKI to convert raw confidence values to belief functions is described in appendix B; of course, the conversion method described there is only one possible method that could be used to convert the measurements into belief functions.

In this chapter the two components of the evidence generation process will be explored. The first section of this chapter addresses the generation of initial labels based on the compatibility of data-elements with model-elements. Generation of the initial labels for elements on the edge-level and face-level is discussed in detail. The second portion of the chapter addresses the process of generating updating evidence for an element's label based on the compatibility between its label and its siblings' labels,

6.1. Computing Initial Belief Functions for Data Elements

As described in chapter 5, an image-element's initial label and belief function are obtained by checking constraints between the element itself and elements on the model panel

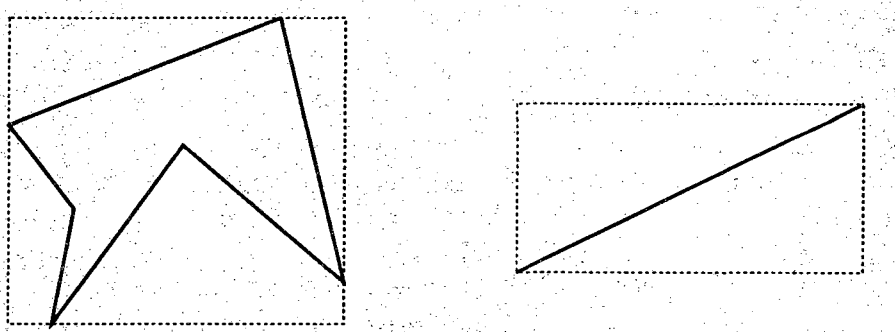
of the blackboard. Obviously, the constraints used to provide initial evidence need not be the same for all levels of the hierarchy. The evidence generated by measuring the degree to which the constraints are met only needs to focus on members of an element's FOD or their compliments since these are the only subsets of Θ that can be used by the evidence accumulation scheme described in chapter 5. The output of the metrics must range from 0.0 to 1.0 in order to use the technique presented in appendix B to convert the measurements into bpa's.

Before any labels may be generated for an element, its FOD must be determined. If an element has a parent, then its FOD is defined to be the children of its parent's label-element, as described in chapter 5. For example, consider Fig. 5.3. If edges $\{E_1, ..., E_8\}$ on the data panel are grouped into face F_1 and F_1 is matched with F_A, then the FOD for each edge in the group would be

$$\Theta = \{E_A, E_B, E_C, E_D\}$$

Note that, since label information of elements on upper levels of the blackboard is used to determine the FOD for an element's label, it is advantageous to determine the labels of elements on the higher levels first, and then work down to elements on lower levels.

If an element has not been placed into a group and, therefore, has no parent, then a different tack must be taken to form its FOD. In this case, the *extents* of the elements are used to determine their FODs. The term extent is taken from the computer graphics arena [FolVan82] and is defined to be the minimum-size rectangle with edges parallel to the coordinate axis that contains an object. Examples of the extents for an edge and a face are shown in Fig. 6.1.



(a) (b)

FIGURE 6.1 This figure shows the extent for an arbitrary face and an arbitrary edge. The objects in this figure are drawn using solid lines and their extents are the dashed boxes.

The FOD for an orphan face-element includes any model element whose extent overlaps its own. Fig. 6.2 demonstrates the process of determining an orphan face's FOD. In cases (b) and (c) of this figure, F_A would be placed in F_1's FOD; however, in case (a) it would be excluded

from the FOD because the two extents do not overlap.

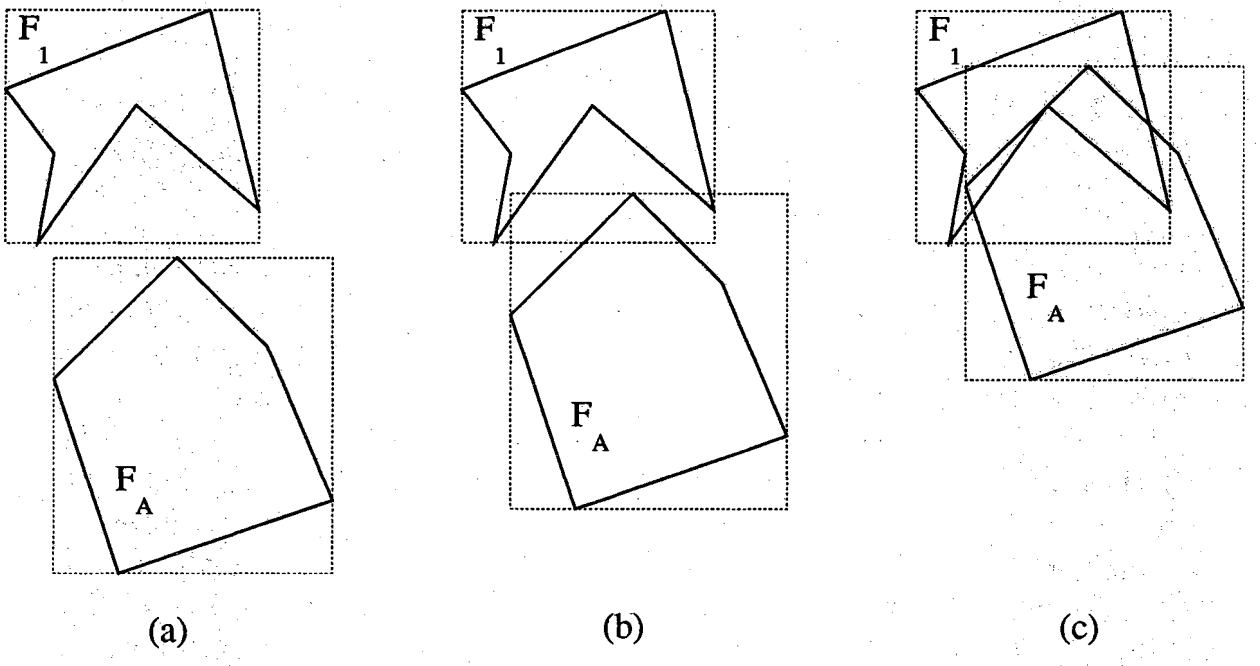


FIGURE 6.2 This figure demonstrates how extents are used to determine a face element's FOD. In this figure, F_A would be placed in F_1's FOD in cases (b) and (c) because their extents (shown as dashed boxes) overlap. Conversely, it would be excluded in case (a) because the two extents do not overlap.

A similar method is used to determine the FODs for orphan edges. However, the method must be modified slightly because two edges can be arbitrarily close and not have overlapping extents (for example, if they are both parallel to the same coordinate axis). To guarantee that all model edges are included in an edge's FOD that should be, the extent of the edge is expanded. Fig. 6.3. shows how the extent of an edge is expanded by adding a border around the extent. The size of the border around the edge's extent, D_{max}, is set by the user and reflects the maximum expected misregistration between the image and the expected scene. Fig. 6.4 demonstrates the process used to determine if a model edge is included in an orphan edge's FOD. In panel (a) of this figure, the model edge would not be included in the edge's FOD because the extents do not overlap. However, the model edge would be included on the FOD in cases (b) and (c) because the extents overlap. Note that although the image elements and the model elements are on different panels of the blackboard, it is possible to speak of distances and angles between them because they are both projected into the same world coordinate system.

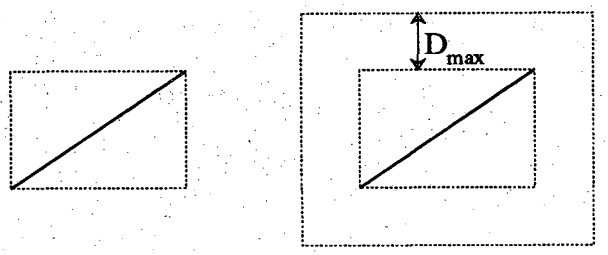


FIGURE 6.3 This figure demonstrates how a border is added to an edge's extent. Panel (a) shows the edge's original extent; panel (b) shows the edge's expanded extent.

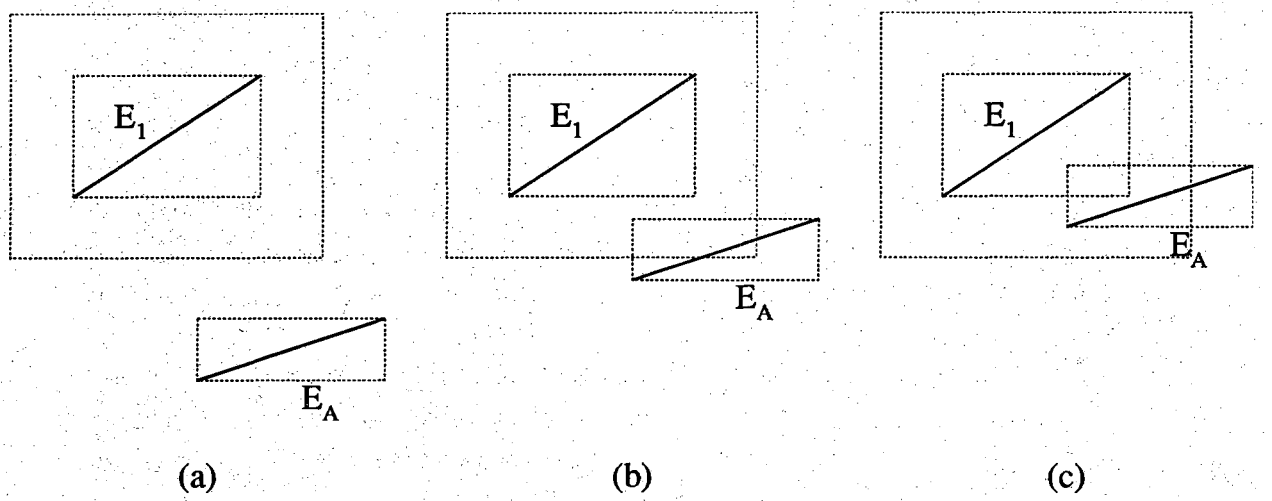


FIGURE 6.4 This figure demonstrates how extents are used to determine an edge element's FOD. In this figure, E_A would be placed in E_1's FOD in cases (b) and (c) because model elements' extents (shown as dashed boxes) overlap with the data edge's expanded extent. Conversely, it would be excluded in case (a) because the two extents do not overlap.

6.1.1. Computing Initial Belief Functions for Edge-Elements

When determining initial matches between edges, PSEIKI's labeler KS tries to match a data edge with the edge in its FOD that lies closest to the same line. To find the match partner of a data edge, the KS measures the degree of "collinearity" between the edge and all the model edges in its FOD; it then chooses as the match partner the model edge with which the data edge is most collinear. The belief of the match made then is set to the degree of collinearity between the two edges.

The following formula is used as the measure of collinearity between an edge detected in the image and an edge from the expected scene (edge E_i is the model edge and E_j is the data edge).

$$\text{ES\_collinearity}(E_i, E_j) = \frac{D_{max} - D_{perp}}{D_{max}} \times \frac{D_{max} - D_{par}}{D_{max}} \times \cos(\theta)$$

where D_{perp} is the distance from the middle of E_1 to the line defined by E_A, D_{par} is the misregistration along the direction E_A, D_{max} is the maximum allowable value for either of the two misregistrations, and θ is the acute angle between the segments (see Fig. 6.5). The value for D_{max} reflects the maximum expected misregistration between the image and the expected scene and is set equal to the amount that the edges' extents are expanded when their FODs are determined.

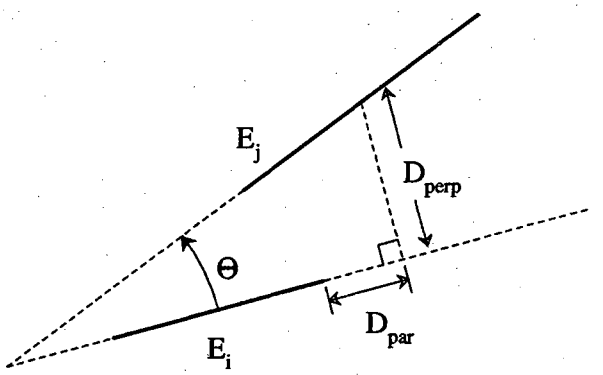


FIGURE 6.5 This figure shows parameters used in the definition of collinearity.

To determine an edge-element's label, PSEIKI's labeler KS computes its bpa over Θ by applying the ES_collinearity measure to each element in its FOD. For example, if edge E_1's FOD was determined to be

$$\Theta = \{E_A, E_B, E_C, E_D\}$$

then the formula might produce the following ES_collinearity measurements,

$$\text{ES\_collinearity}(E_A, E_1) = 0.43$$

$$\text{ES\_collinearity}(E_B, E_1) = 0.11$$

$$\text{ES\_collinearity}(E_C, E_1) = 0.73$$

$$\text{ES\_collinearity}(E_D, E_1) = 0.56$$

Using the technique described in appendix B, the ES_collinearity measurements can be converted to the following bpa for E_1 by normalizing all the values by the total confidence.

$$m_{E_1}(E_A) = \text{ES\_collinearity}(E_A, E_1) \,/\, \text{total\_confidence} = 0.24$$

$$m_{E_1}(E_B) = \text{ES\_collinearity}(E_B, E_1) \,/\, \text{total\_confidence} = 0.06$$

$$m_{E_1}(E_C) = \text{ES\_collinearity}(E_C, E_1) \,/\, \text{total\_confidence} = 0.40$$

$$m_{E_1}(E_D) = \text{ES\_collinearity}(E_D, E_1) \,/\, \text{total\_confidence} = 0.30$$

$m_{E_1}(\cdot) = 0.0$ for all other subsets of Θ

where total_confidence $= 0.43 + 0.11 + 0.73 + 0.56 = 1.83$

With this bpa, E_1's label would be set to E_C with a belief of 0.40.

Another procedure is used to initialize the bpa if the ES_collinearity measures sum to less than one. Assume for a moment that the distance cutoff, D_{max}, is decreased resulting in the following ES_collinearity measurements

ES_collinearity(E_A, E_1) = 0.23

ES_collinearity(E_B, E_1) = 0.11

ES_collinearity(E_C, E_1) = 0.13

ES_collinearity(E_D, E_1) = 0.26

In this case, the amount of confidence left uncommitted by the metric, 0.27, is considered to be the amount of ignorance in the identity of edge E_1 and is set to be the probability mass of the FOD, Θ. We set the probability mass in Θ to the uncommitted confidence because ES_compatibility(E_i, E_1) measures the belief that the edge E_1's identity is E_i. Clearly, if the edge's identity cannot be determined to be any of the elements in its FOD with a sufficiently high degree of confidence, then some belief about its identity should be left uncommitted. Using this procedure, the following bpa for E_1 is constructed.

$m_{E_1}(E_A)$ = ES_collinearity(E_A, E_1) = 0.23

$m_{E_1}(E_B)$ = ES_collinearity(E_B, E_1) = 0.11

$m_{E_1}(E_C)$ = ES_collinearity(E_C, E_1) = 0.13

$m_{E_1}(E_D)$ = ES_collinearity(E_D, E_1) = 0.26

$m_{E_1}(\Theta) = 1.0 - \text{total\_confidence} = 0.27$

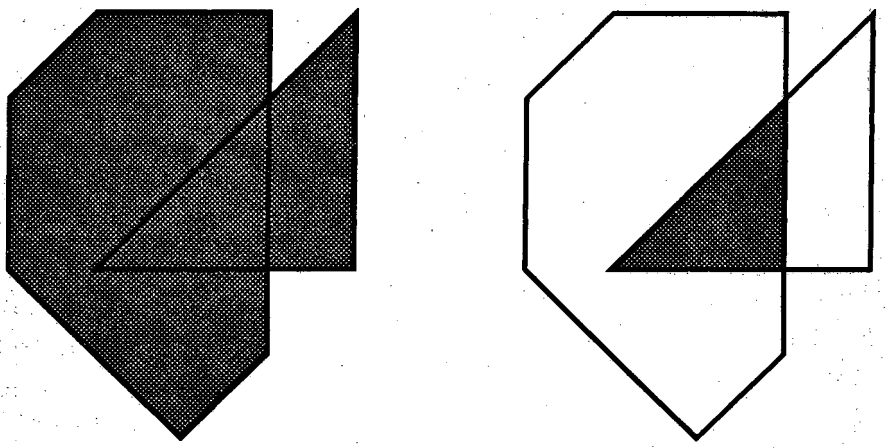
$m_{E_1}(\cdot) = 0.0$ for all other subsets of Θ

where total_confidence $= 0.23 + 0.11 + 0.13 + 0.26 = 0.73$

6.1.2. Computing Initial Belief Functions for Face-Elements

When determining initial matches between face-elements, PSEIKI's labeler KS tries to maximize the percentage of overlap between matched elements. That is, to determine a face-element's label, the percent of overlap between it and all elements of its FOD is measured and the model face-element with maximum overlap is selected. The percentage of overlap between two face-elements is defined to be the area of their intersection divided by the area of their union. This notion is shown in Fig. 6.6 and can be expressed as

$$ES\_overlap(F_{model}, F_{image}) = \frac{Area_{intersection}}{Area_{union}}$$



(a) Union (b) Intersection

FIGURE 6.6 This figure shows the union and intersection of two faces.

To improve the computational efficiency, the percent of overlap between two faces currently is approximated by the percentage of overlap of their two extents.

6.2. Computing Updating Belief Functions for Data Elements

After the initial matches are established between elements, the labeler KS provides evidence about the validity of the an element's label based on the label's consistency with the element's siblings' labels. In general, two metrics are required for updating an element's label belief function. The two metrics must provide measures of *compatibility* and *incompatibility* between the element and its siblings. The compatibility metric is used to provide confirmatory evidence that the element's label is correct. Conversely, the incompatibility metric provides disconfirmatory evidence about the element's label. Both metrics should range between 0.0 and 1.0 to facilitate conversion of their values to an updating bpa. It is illustrative to examine how one element can be used to update the belief in another element's label when both have the same label. When this process is understood, the case in which two elements have different labels follows naturally.

6.2.1. Computing Updating Belief Functions for Edge-Elements with the Same Label

Collinearity(·) and **noncollinearity(·)** are the metrics used to determine the (in)compatibility between two edges with the same label (the collinearity metric is related to the ES_collinearity measure used to establish initial matches). That is, if E_i and E_j are edges

in the data panel and have the same label, then collinearity(E_i, E_j) is the measure of compatibility between them. Collinearity is defined as

$$\text{collinearity}(E_i, E_j) = \frac{D_{max} - D_{perp}}{D_{max}} \times \cos(\theta)$$

where θ is the acute angle between the two edges and D_{perp} the distance from the middle of E_j to the line defining E_i (see Fig. 6.5). D_{max}, the maximum allowable value for D_{perp}, is a user-specified heuristic parameter or function. For the computation of updating evidence, D_{max} is set in a manner different from that described in Section 6.1.1; its value is set equal to the length of E_i. Setting D_{max} in this manner is justified by the rationale that the maximum allowable distance between two data-elements with the same label should be a function of the sizes of the data-elements.

Likewise, the incompatibility between two edges, can be measured by calculating the **noncollinearity**(E_i, E_j) between them. Noncollinearity is defined as

$$\text{noncollinearity}(E_i, E_j) = \frac{D_{perp}}{D_{max}} \times \text{scale}(E_i) \times \sin(\theta)$$

where scale(E_i) depends on the length of E_1*.

Because the (in)compatibility measures are defined heuristically, it usually is advantageous to limit the amount of evidence that they can provide. This is accomplished by scaling the measures by a level-specific scale factor SF ($0.0 \leq SF \leq 1.0$). Thus the (in)compatibility measures for the edge-level can be defined as:

$$\text{compatibility}(E_i, E_j) = \text{collinearity}(E_i, E_j) \times SF_{edge}$$

$$\text{incompatibility}(E_i, E_j) = \text{noncollinearity}(E_i, E_j) \times SF_{edge}$$

Once the (in)compatibility between the two edges has been determined, the technique described in appendix B can be used to convert them into a bpa. For example, assume that E_1 and E_2 exhibit maximal beliefs for the same model edge, E_A, and that the labeler KS is using E_2 to update the belief of E_1's label. To do so, the labeler measures the collinearity and the noncollinearity of E_1 and E_2, If the results of the (in)compatibility measurements are

$$\text{compatibility}(E_2, E_1) = 0.8$$

$$\text{incompatibility}(E_2, E_1) = 0.1$$

the belief in E_2's label (say, for example, 0.8) can be used to create an "updating" confidence function for E_1 as follows:

*The scale factor is provided to limit the amount of disconfirmatory evidence generated by small edges which may be due to noise.

$$\begin{aligned} Conf_{2\rightarrow 1}(\{E_A\}) &= m_2(\{E_A\}) \times \text{compatibility}(E_2, E_1) \\ &= 0.64 \end{aligned}$$

$$\begin{aligned} Conf_{2\rightarrow 1}(\{\neg E_A\}) &= m_{E_2}(\{E_A\}) \times \text{incompatibility}(E_2, E_1) \\ &= 0.08 \end{aligned}$$

Since the confidence function has some belief left uncommitted, E_1's updating bpa can be defined as

$$m_{\underset{2\rightarrow 1}{update}}(\{E_A\}) = Conf_{2\rightarrow 1}(\{E_A\}) = 0.64$$

$$m_{\underset{2\rightarrow 1}{update}}(\{\neg E_A\}) = Conf_{2\rightarrow 1}(\{\neg E_A\}) = 0.08$$

$$m_{\underset{2\rightarrow 1}{update}}(\{\Theta\}) = 1.0 - Conf_{2\rightarrow 1}(\{E_A\}) - Conf_{2\rightarrow 1}(\{\neg E_A\}) = 0.28$$

where the probability mass for the FOD was set to the uncommitted portion of belief.

6.2.2. Computing Updating Belief Functions for Face-Elements with the Same Label

The (in)compatibility metrics for face-elements are called **colocate(·)** and **noncolocate(·)**. These two metrics are designed to measure how close two face-elements are to each other by measuring the distance between their centroids. The compatibility metric between two face-elements, colocate(F_1, F_2), is defined as

$$\text{colocate}(F_1, F_2) = \frac{D_{max} - D_{centroid}}{D_{max}}$$

where $D_{centroid}$ is the distance between the centroids of the two faces. D_{max} is the maximum allowable value for $D_{centroid}$; currently, it is set to the length of the diagonal of F_1's extent. Again, this is done to scale, by an element's size, the evidence that the metric can provide. To improve computational efficiency, the centroid of a face currently is approximated by the centroid of its extent. Similarly, noncolocate(F_1, F_2), the face-level incompatibility metric, is defined as

$$\text{noncolocate}(F_1, F_2) = \frac{D_{centroid}}{D_{max}}$$

Note that these metrics can be used for range data by extending the definitions to use the directions of the normal vectors of the two faces. In the three dimensional case, these metrics could be defined as

$$\text{colocate}_{3D}(F_1, F_2) = \frac{D_{max} - D_{centroid}}{D_{max}} \times \cos(\theta)$$

and

$$\text{noncolocate}_{3D}(F_1, F_2) = \frac{D_{centroid}}{D_{max}} \times \sin(\theta)$$

where the distance parameters are defined as before and θ is the acute angle between the two normal vectors. This extension is not needed currently because of the two-dimensional aspect of the mobile robotic environment.

Another metric that can be used to compute the incompatibility between two faces, F_i and F_j, on the data panel of the blackboard, is the fraction of overlap between them, **overlap**(F_i, F_j). To understand how this metric is used, consider the following example. Assume that the incompatibility of two faces on the data panel, F_1 and F_2, is being computed and that the two are thought to correspond with two non-overlapping faces on the data panel, F_A and F_B, respectively. If faces F_1 and F_2 overlap by 10%, then the incompatibility between them can be defined to be

$$\text{incompatibility}(F_1, F_2) = \text{overlap}(F_1, F_2) = 0.1$$

6.2.3. Computing Updating Belief Functions for Elements with Different Labels

If two elements correspond to different model-elements, a rigid motion transformation is applied to one of them before the computation of the (in)compatibility metrics. This has the effect of enforcing relational constraints between the two data-elements. For example, if edges E_1 and E_3 are thought to correspond to model edges E_A and E_B, respectively, then the measure of compatibility between E_1 and E_3 would be defined as

$$\text{compatibility}(E_3, E_1) = \text{collinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{edge}$$

where $T_{E_A \rightarrow E_B}$ is the rigid motion transformation that makes model edge E_A collinear with model edge E_B.

Fig. 6.7 can be used to aid in the explanation of how the transformation is defined. First, for a given pair of non-parallel edges, the vertices on the convergent and the divergent sides of the edges are distinguished; the convergent side of the two edges is the side on which they would meet if extended. The transformation $T_{E_A \rightarrow E_B}$ is accomplished by rotating edge E_A about its convergent vertex through an angle that makes the edges parallel; subsequently, E_A is translated so that the two convergent vertices coincide. Performing this transformation forces model-elements to be compatible; in other words,

$$\text{collinearity}(E_B, T_{E_A \rightarrow E_B}(E_A)) = 1.0$$

Note that the definition of the transformation is not well defined. There are two transformations that can be used to make the two model edges collinear depending on the direction

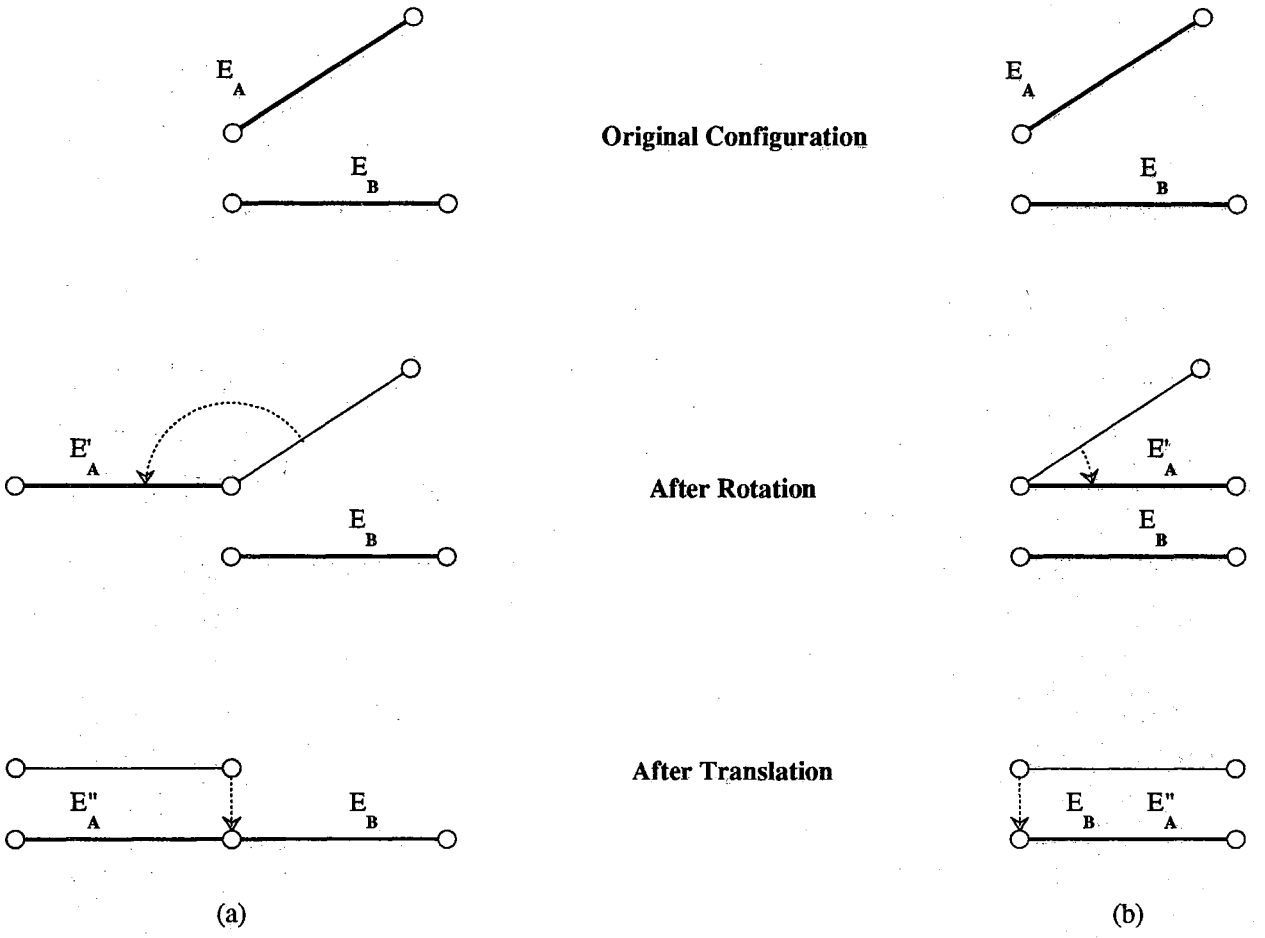


FIGURE 6.7 This figure shows the rigid motion transformation that makes two model-elements collinear. Panel (a) shows the transformation created by "unfolding" the two edges. Panel (b) shows the transformation created by "collapsing" the two edges.

that edge E_A is rotated. The first transformation "unfolds" the two model edges by forcing the angle between them to be 180 degrees; this type of transformation is shown in panel (a) of Fig. 6.7. The other type of transformation "collapses" the two edges onto each other by forcing the angle between them to be 0 degrees. It is impossible to determine completely from the geometry of the model edges which transformation will be needed to make two data edges collinear; the transformation also depends on the direction that the image is misregistered from the expected scene. Therefore, the transformation that should be used to make two edges collinear must be determined at runtime. PSEIKI's labeler KS computes the collinearity of the two edges using both transformations and uses the transformation that results in the largest collinearity measurement. The same transformation is then used to determine the incompatibility of the two edges.

Consider, as an example, how the relational constraints are checked by transforming elements and measuring their (in)compatibility. Assume that edge E_3 is being used to provide

updating evidence about the label of edge E_1. Furthermore, assume that edge E_1 has label E_A and edge E_3 has label E_B. To measure the extent to which the geometrical relationship between E_1 and E_3 is the same as the one between E_A and E_B, the labeler carries out the following (in)compatibility computations:

$$\text{compatibility}(E_3, E_1) = \text{collinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{edge}$$

$$\text{incompatibility}(E_3, E_1) = \text{noncollinearity}(E_3, T_{E_A \rightarrow E_B}(E_1)) \times SF_{edge}$$

where $T_{E_A \rightarrow E_B}$ is the transformation that makes the model edges E_A and E_B coincident and results in the greatest measured collinearity between E_3 and the transformed version of edge E_1. Clearly, compatibility(E_3, E_1) = 1.0 implies that the geometrical relationship between E_1 and E_3 in the data is exactly the same as between E_A and E_B in the model (in this case, incompatibility(E_3, E_1) = 0.0). If the compatibility calculations yielded the following results:

$$\text{compatibility}(E_3, E_1) = 0.7$$

$$\text{incompatibility}(E_3, E_1) = 0.4$$

and the belief in E_3's label was 0.95 then the following confidence function could be defined by using the (in)compatibility measures and the belief in E_3's label.

$$\begin{aligned} Conf_{3 \rightarrow 1}(\{E_A\}) &= m_{E_3}(\{E_B\}) \text{ x compatibility}(E_3, E_1) \\ &= 0.7 \text{ x } 0.95 \\ &= 0.665 \end{aligned}$$

$$\begin{aligned} Conf_{3 \rightarrow 1}(\{\neg E_A\}) &= m_{E_3}(\{E_B\}) \text{ x incompatibility}(E_3, E_1) \\ &= 0.4 \text{ x } 0.95 \\ &= 0.38 \end{aligned}$$

Since the confidence is overspecified, the updating bpa can be defined by normalizing with the total confidence.

$$m^{update}_{3 \rightarrow 1}(\{E_A\}) = 0.64$$

$$m^{update}_{3 \rightarrow 1}(\{\neg E_A\}) = 0.36$$

The same technique of checking relational constraints can be used on the elements residing on the face level. That is, the (in)compatibility between face elements can be measured by applying the (non)colocate metrics to transformed face elements with different labels. However, since the metrics used to calculate the (in)compatibility between face-elements use only the distance between centroids for their computations, only a translational transformation is required. Formally, the transformation $T_{F_i \rightarrow F_j}$ merely translates F_i's centroid until it is coincident with F_j's centroid. The transformation used to measure face-level relational constraints

is shown in Fig. 6.8.

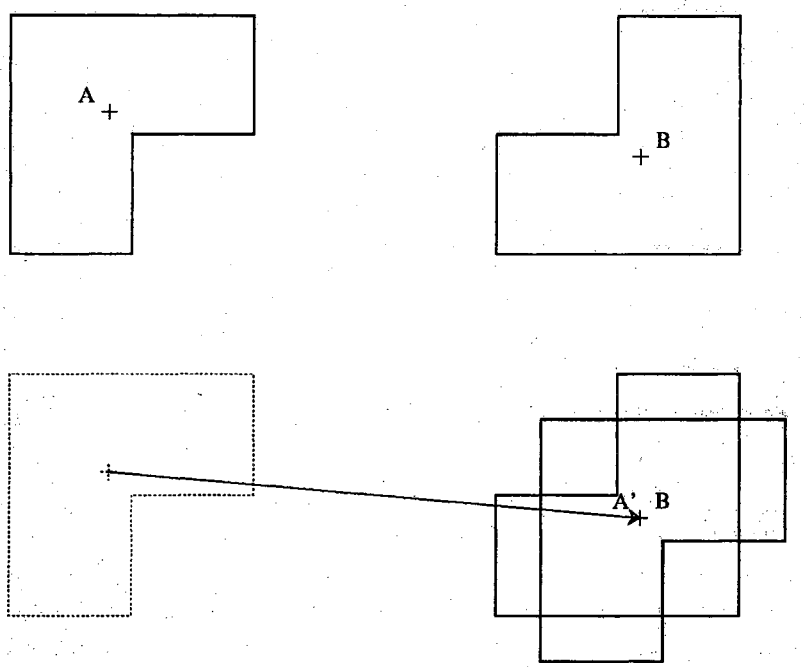


FIGURE 6.8 This figure shows the rigid motion transformation that makes the centroid face F_A coincident with the centroid of the transformed version of face F_B. The crosses inside each face indicate the location of its centroid.

Note that if the metrics are extended to work in three-space, as previously discussed, then there should be a rotational component to the transformation that would make the faces' normal vectors collinear.

In reality, a single procedure is used for enforcing both the local and the relational constraints within a group. Note that if the identity transformation, $T_{E_X \rightarrow E_X}$ is used, the (in)compatibility calculations for relational constraints reduce to the computations required for (in)compatibility calculations for mutual consistency in Sections 6.2.1 and 6.2.2.

To make the concepts introduced in this chapter more concrete, we will show an example of the how face elements are labeled and how the belief in those labels are updated. In this example, assume that the expected scene consists of a single object with four faces, as shown in the left panel of Fig. 6.9. Also assume that a region-based preprocessor presented PSEIKI with the observed scene depicted in the right panel of Fig. 6.9.

The first step in the labeling process consists of determining the frames of discernment for the faces on the data panel. As previously described, a model element is include in a face-level data element's FOD if the extents of the two elements overlap. For example, if F_A was the only model face whose extent overlapped with face F_1's extent, then F_1's FOD would consist entirely of $\Theta_{F_1} = \{F_A\}$. On the other hand, if the extent of face F_6 overlapped with the

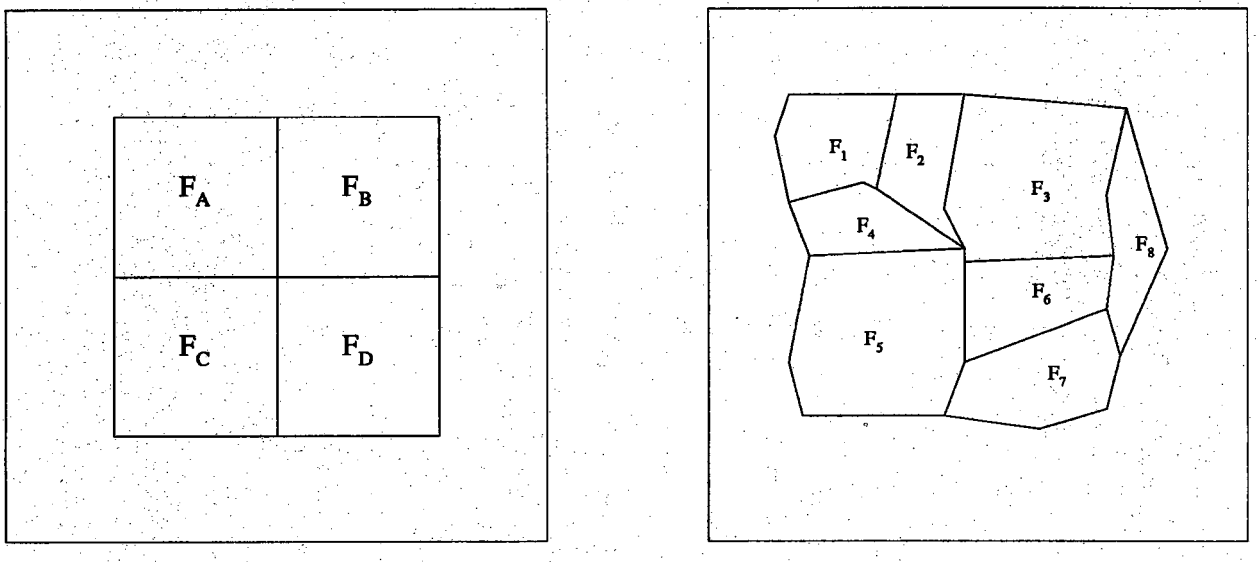


FIGURE 6.9 The left and right panels of this figure show the model and data panels of the blackboard, respectively. This figure is used in the example in text which describes how the labeler KS initializes and updates the belief in the labels of face-level elements.

extents all of the model faces, then its FOD would be $\Theta_{F_6} = \{F_A, F_B, F_C, F_D\}$.

After the initial FODs for the face elements have been determined, the belief function of each element is initialized by measuring the percentage that the face's extent overlaps with the extent of each model element in its FOD. For example, the following probability masses could result from measuring the percentage of overlap between face F_6 and the model faces.

$$\text{ES\_overlap}(F_A, F_6) = \frac{F_1 \cap F_A}{F_1 \cup F_A} = 0.05$$

$$\text{ES\_overlap}(F_B, F_6) = \frac{F_1 \cap F_B}{F_1 \cup F_B} = 0.1$$

$$\text{ES\_overlap}(F_C, F_6) = \frac{F_1 \cap F_C}{F_1 \cup F_C} = 0.1$$

$$\text{ES\_overlap}(F_D, F_6) = \frac{F_1 \cap F_D}{F_1 \cup F_D} = 0.35$$

The following probability masses are obtained by using the process described in appendix B.

$$m_{F_6}\{F_A\} = \text{ES\_overlap}(F_A, F_6) = 0.05$$

$$m_{F_6}\{F_B\} = \text{ES\_overlap}(F_B, F_6) = 0.1$$

$$m_{F_6}\{F_C\} = \text{ES\_overlap}(F_C, F_6) = 0.1$$

$$m_{F_6}\{F_D\} = \text{ES\_overlap}(F_D, F_6) = 0.35$$

$$m_{F_6}\{\Theta_{F_6}\} = 1.0 - 0.05 - 0.1 - 0.1 - 0.35 = 0.4$$

$$m_{F_6}(\cdot) = 0.0 \quad \text{for all other subsets of } \Theta_{F_6}$$

Thus face F_6 would be assigned label F_D with belief 0.35. The same process is used to initialize the belief functions of the other face elements on the data panel. Assume for the example that the other faces received the following labels.

Face	Label	Belief
F_1	F_A	0.30
F_2	F_A	0.42
F_3	F_B	0.72
F_4	F_A	0.33
F_5	F_C	0.67
F_6	F_D	0.26
F_7	F_D	0.31
F_8	F_B	0.20

After each face's belief function has been initialized, the grouper KS is allowed to group compatible faces into objects. If we assume that one of the groups formed by the grouper KS consists of faces F_1, ..., F_7, then these faces can be used to update the belief in each other's labels. For our example, we will concentrate on the process used to update the belief in the label of face F_6. For each face in the group, excluding face F_6, we measure the (in)compatibility of the face with face F_6 using the *colocate()* and *noncolocate()* metrics and appropriate transformations. For example, since F_6 and F_7 have the same label, the updating evidence provided by measuring their consistency is computed as follows (assuming that SF_{face} is equal to 1.0)

$$m_{\underset{7\to6}{update}}(\{F_D\}) = m_{F_7}(\{F_D\}) \times \text{colocate}(F_7, F_6) \times SF_{face}$$

$$= 0.31 \times 0.4 \times 1.0$$

$$= 0.13$$

$$m_{\underset{7\to6}{update}}(\{\neg F_D\}) = m_{F_7}(\{F_D\}) \times noncolocate(F_7, F_6) \times SF_{face}$$

$$= 0.31 \times 0.6 \times 1.0$$

$$= 0.18$$

$$m_{\underset{7\to6}{update}}(\{\Theta_{F_6}\}) = 1.0 - 0.13 - 0.18$$

$$= 0.69$$

However, since the other faces in the group do not have the label F_D, face F_6 must be transformed before the metrics are applied. For example, the updating evidence for face F_6's label generated by checking its consistency with the label of face F_5 can be computed as

$$m_{\underset{5\to6}{update}}(\{F_D\}) = m_{F_5}(\{F_D\}) \times colocate(F_5, T_{F_D\to F_C}(F_6)) \times SF_{face}$$

$$= 0.67 \times 0.8 \times 1.0$$

$$= 0.53$$

$$m_{\underset{5\to6}{update}}(\{\neg F_D\}) = m_{F_5}(\{F_D\}) \times noncolocate(F_5, T_{F_D\to F_C}(F_6)) \times SF_{face}$$

$$= 0.67 \times 0.2 \times 1.0$$

$$0.14$$

$$m_{\underset{5\to6}{update}}(\{\Theta_{F_6}\}) = 1.0 - 0.53 - 0.14$$

$$0.33$$

Updating evidence can be generated by checking face F_6's consistency with the other faces in the group in a similar manner. After the all of the faces in the group have been used to provide evidence on the validity of face F_6's label, the resulting updating bpa is combined with F_6's bpa using Barnett's formulas to yield a new belief function.

Note that, in this example, we have not addressed the effects other KSs would have on the processing. For example, the merger KS would most likely merge the following groups of faces at some point in the processing because the elements in each group are adjacent and have the same label.

$\{F_1, F_2, F_4\} \to F_9$

$\{F_6, F_7\} \to F_{10}$

The composite faces formed by the merger would then be labeled and updated in the manner described above. In the next chapter, we will describe the methods used by the splitter, merger and grouper KSs to create and modify groups.

CHAPTER 7

EVIDENTIAL ASPECTS OF THE GROUPER, SPLITTER AND MERGER KNOWLEDGE SOURCES

PSEIKI's low-level preprocessors produce data only for the lower levels of the blackboard; thus, the system needs to generate data elements on higher levels. Furthermore, data presented to PSEIKI by its low-level preprocessors is often far from optimal. Many times, image structures that should remain separate are merged into a single structure (i.e. the image is undersegmented) or a structure is incorrectly broken into a number of smaller ones (i.e. the image is oversegmented). In fact, it is common for a single image to be undersegmented in one section and oversegmented in another. The grouper, splitter and merger KSs are designed to compensate for these deficiencies by building objects on upper levels of the blackboard from elements on lower levels and by correcting segmentation errors.

It is the grouper KS's task to create data-elements on the upper levels of the hierarchy by forming groups of elements on lower levels. Many previous systems that performed element grouping used perceptual organization principles descended from Gestalt Theory [Koh47]. Gestalt theory is a psychological tenet which states that perception occurs as a whole process not the combination of a number of more elemental processes. One of the main products of the Gestalt school was a catalog of a large number of phenomena that produced perceptual grouping. Fig. 7.1 shows some of the grouping phenomena categorized by the Gestaltists. Although this initial thrust into perceptual organization offers little help to computer vision systems, some vision systems are able to discover perceptual groups based on the related principles of "transformations" [WitTen83a], [WitTen83b] and "interestingness" [LawMcC87]. The use of perceptual grouping in computer vision systems is also discussed in [Mar82], [Low85].

The splitter and merger KSs are designed to correct grouping errors produced by the low-level preprocessors, by the grouper KS and by each other. The merger KS tries to correct oversegmented images by merging elements on one level of the blackboard into a single group on the same level. The splitter KS's task is to break an element into smaller elements all of which reside on the same level of the blackboard as the original element. This splitting is done to correct an undersegmented image. These two KSs use many of the classic splitting and merging techniques described in [BriFen70], [HorPav74], [Zuc76], and particularly those expressed as rules in [NazLev84].

7.1. The Grouper Knowledge Source

The grouper KS builds data elements on the upper levels of the hierarchy from data elements deposited by the low-level vision system. It does this in a data-driven manner by

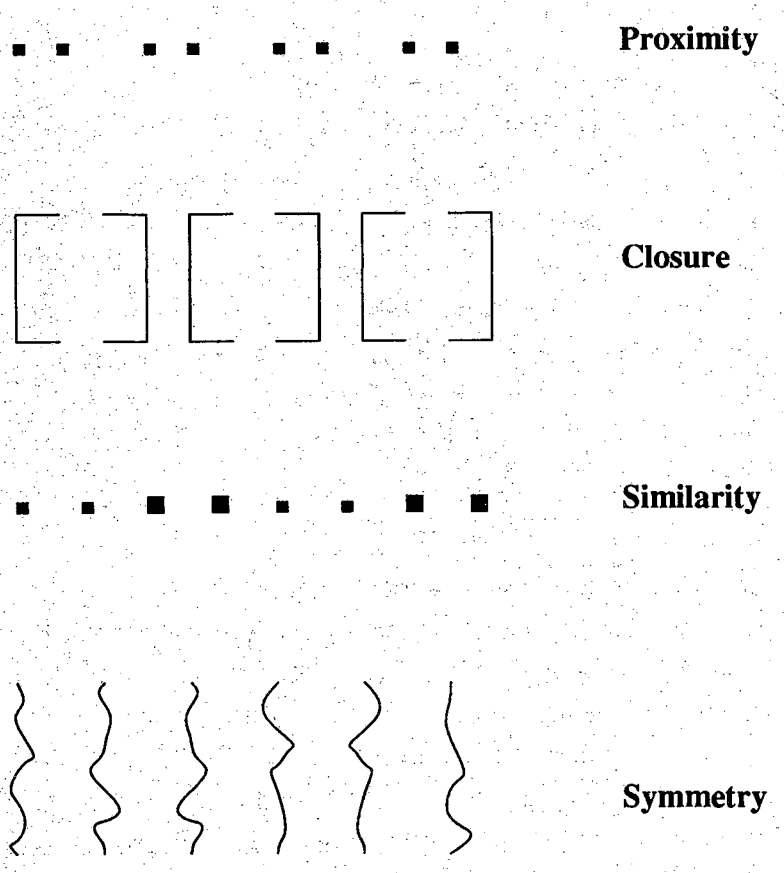


FIGURE 7.1 These are some examples of the grouping phenomena cataloged by the Gestaltists.

grouping objects on the lower levels of the hierarchy into progressively higher levels. For example, if an edge-based preprocessor is used to generate input data, the grouper first groups edge-elements into faces and then groups the faces into objects, and so on.

Fig. 7.2 shows a simple example of how the grouping is performed; panel (a) shows the expected scene, panel (b) shows the edges presented to PSEIKI by an edge-based preprocessor, and panel (c) shows the initial labels for those edges.* The grouper KS is triggered by the monitor when the monitor detects an element on the data panel that has no parents. These orphan elements can have a number of origins: The low-level preprocessor deposits a large number of orphan elements onto the data panel at the beginning of processing; in fact, all edge-level elements deposited by an edge-based preprocessor are orphans, as are all face-level elements deposited by a region-based preprocessor. Any data element created by the grouper KS, splitter KS or merger KS also is an orphan initially. When the KS is triggered by the monitor, a knowledge source activation record (KSAR) is built indicating that the orphan element

*The labels shown in Fig. 7.2 are intended only for the purpose of explanation here. In actual practice, even for simple imagery, the initial label map may be much more chaotic, depending upon the extent to which an image is degraded by noise and other artifacts.

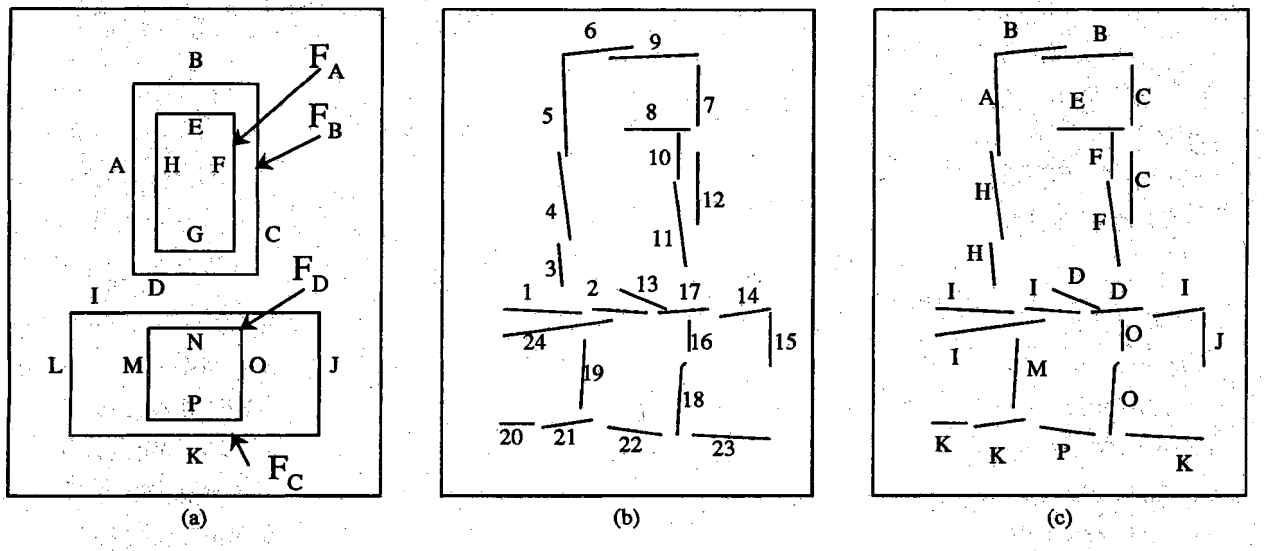


FIGURE 7.2 This figure shows an expected scene in panel (a), the edges produced by an edge-based preprocessor in panel (b), and their initial labels in panel (c).

should be used as a *seed-element* of a group. For example, if we assume that all of the edges shown in Fig. 7.2(b) are orphans deposited on the data panel by an edge-based preprocessor at the start of processing, then a KSAR is built for each edge indicating that it should be grouped. After the monitor triggers the KS by building the KSAR, it is up to the scheduler to determine when the KS will fire and form the specified group.

The scheduler fires the grouper KS when a new data element with a particular label is needed. At this point, the scheduler determines all of the grouper KSARs whose seed-elements can be a child of an element with the desired label and ranks them based on the elements' belief, size and strength. For example, in Fig. 7.2, at some point in the course of blackboard processing, a new data element with label F_D may be desired. To form an element with this label, the scheduler would rank the grouper KSARs for edge elements with labels E_M, E_N, E_O and E_P, because these are the only elements that could be the children of a face with label F_D. The scheduler then chooses the highest ranked KSAR and fires the grouper KS. When the grouper KS is fired, it creates a parent-element one level up on the blackboard from the seed-element with the seed-element as the parent's only child. It then determines the set of all elements that could possibly be the siblings of the seed-element, based on their labels. In the example, suppose that edge E_{19} was chosen as the seed-element, then the only edges that could possibly become its siblings are edges E_{16}, E_{18} and E_{22} because these are the only edges whose model elements are siblings of edge E_{19}'s model element. After the set of candidate siblings have been determined, the (in)compatibility metrics discussed in chapter 6 are used to determine which candidates get grouped with the seed element. A candidate element will be grouped with the seed-element only if the compatibility metric yields a value above a user-specified threshold. For example, if the compatibility threshold has been set to 0.5 and the

following compatibility measurements were made

$$\text{collinearity}(E_{16}, T_{E_M \rightarrow E_O}(E_{19})) = 0.65$$

$$\text{collinearity}(E_{18}, T_{E_M \rightarrow E_O}(E_{19})) = 0.55$$

$$\text{collinearity}(E_{22}, T_{E_M \rightarrow E_P}(E_{19})) = 0.43$$

then the grouper could construct the following initial group of data edges.

$$F_1 = \{E_{16}, E_{18}, E_{19}\}$$

The same process can also be used to find the following initial groups of edges

$$F_2 = \{E_3, E_4, E_8, E_{10}, E_{11}\}$$

$$F_3 = \{E_1, E_2, E_{14}, E_{15}, E_{20}, E_{21}, E_{23}, E_{24}\}$$

$$F_4 = \{E_5, E_6, E_7, E_9, E_{12}\}$$

Note that the blackboard monitor would trigger the grouper KS as soon as these face elements were created because each of them would be an orphan initially. Also note that the grouper KS may incorrectly group some edges into the face. For example, small edges generated by noise may be accidentally included in a group. Also, the grouper may incorrectly include competing elements into a group; two elements are said to compete if they cannot both be present in a consistantly labeled scene interpretation. For example, in F_3, edges E_1 and E_{24} compete with each other. Obviously, the grouper KS should include only one of these competing edges in any group. It is the job of the splitter KS to remove the incorrectly grouped edges from a face. The splitter KS also has the duty to generate multiple faces from a face containing competing edges; the faces that the splitter generates retain only one competing edge at a time. The actions performed by the splitter KS will be explained in greater detail later in the chapter.

The grouper KS groups faces into objects using a similar procedure; however, the grouper uses the *colocate* metric introduced in the last chapter to determine if a candidate face should be grouped with the seed face. We will use the face elements created by the grouper in the last example to explain the processing used by the grouper KS to group face-elements into objects. Assume that labeler KS assigned the following labels and belief values to the above faces.

Face	Label	Belief
F_1	F_D	0.40
F_2	F_A	0.42
F_3	F_C	0.72
F_4	F_B	0.53

If a data element on the object level with label O_A, which is composed of faces F_A and F_B, is desired at some point in the blackboard processing, then the scheduler would rank the

appropriate KSARs based on their face's size and belief values. The scheduler would then fire the grouper KS with the highest ranked KSAR. For this example, assume that the scheduler fired the grouper with F_4 as the seed-element. After the KS is fired, the grouping process proceeds as follows: First, the grouper creates an object level data-element and assigns the seed-element as its only child. It then collects a set of candidate sibling faces based on the their labels. In this example, face F_2 would be the only candidate face because it is the only face with one of the labels, F_A or F_B. If the compatibility threshold was set to 0.5 and the grouper measured the following compatibility measurement

$$\text{colocate}(F_2, T_{F_B \rightarrow F_A}(F_4)) = 0.69$$

then F_2 would be grouped with F_4 to create the following face

$$O_1 = \{F_2, F_4\}$$

7.2. The Merger Knowledge Source

The merger KS also performs a grouping process; however, this process does not build elements on higher levels of the hierarchy from elements on lower levels, as does the grouper KS. Instead, it combines multiple elements on the blackboard into a single, larger element on the same level as the original elements. It combines elements if it is believed that they all can be represented by a single element on the model panel. This combining process can be used to correct grouping errors produced by the low-level preprocessor and the grouper KS. For example, the low-level processor sometimes produces artifacts that break edges into smaller line segments. The merger KS tries to correct this error by joining broken line segments with the same label if they are close together and highly collinear. The merger KS also combines, into a single edge, highly collinear edges that are joined at a degree-two vertex and that have the same label. On the face level, the merger KS will combine two faces with the same label if they are adjacent and grouped in the same object. It will also combine two faces if they have the same label and one completely surrounds the other. Some of the merger KS's actions are shown in Fig 7.3.

The first step in the merging procedure consists of determining if the elements under consideration really need to be merged. For example, it is not feasible for the monitor to check the collinearity of two edges before it builds KSARs to merge them; thus, the merger KS needs to determine if two edges are sufficiently collinear before it merges them. The KS will not merge two edges if the collinearity metric described in chapter 6 yields a value below a user set threshold when applied to the two edges in question. This threshold is usually set to a relatively high value (above 0.75) to keep the KS from merging two edges that should remain separate. For two faces to be merged, it is sufficient that they have the same label, be grouped together and be adjacent. Two faces are said to be adjacent if they contain at least one edge in common.

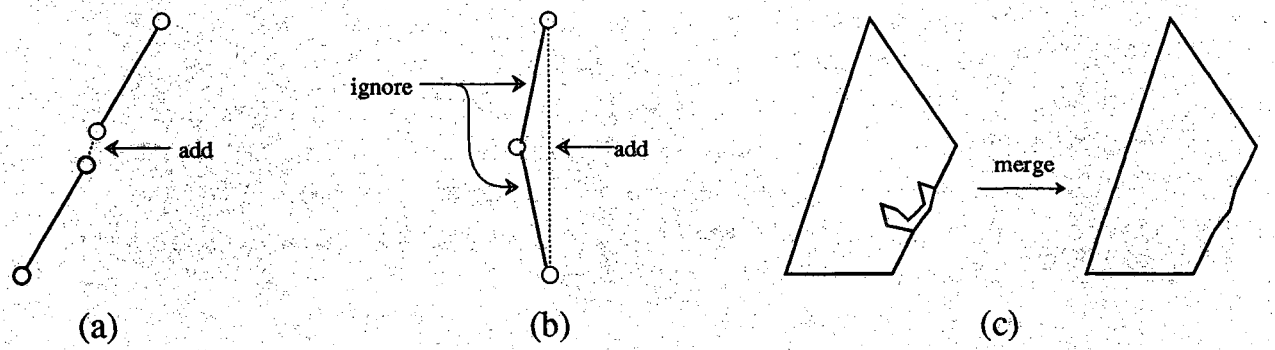


FIGURE 7.3 This figure shows the actions performed by the merger KS. Panel (a) shows how two close, collinear edges can be joined together. Panel (b) demonstrates how two collinear edges can be merged into a single edge. Finally, panel (c) shows how two adjacent face-elements with the same label can be merged if they are grouped together.

Once it has been decided that the elements should be merged, a level-specific procedure is used to merge them. When two edges are to be merged, the KS deposits a new edge element on the blackboard with one vertex from each of the two old edges; these vertices are chosen to give the new edge maximal length. When two faces are to be merged, the merger deposits a new face element on the blackboard whose list of children is the exclusive-or of the lists of the two old edges. That is, an edge is include in the new face's list of children only if it is the child of only one of the old faces. Forming the new face's list of children in this manner prevents the edges that form the border of the two old faces from being included in the new face's list of children. The new element's parameters are also initialized when it is deposited on the blackboard. For example, the strength of a new edge is set to the weighted average of the strengths of the two old edges; likewise, the grey-value of a new face is set to the weighted average of the grey-values of the two old faces. After the new element is created by the merger, any references to both of the old elements is replaced by a reference to the new element. Finally, if the two old elements were always referenced as a pair, a flag is set in the original objects indicating that they should be ignored in further processing; this flag is used because the newly created element superseeds the elements from which it was created.

7.3. The Splitter Knowledge Source

The splitter KS also tries to correct the grouping of incorrectly grouped elements. However, it performs the opposite action of the merger KS; its task is to split data-elements into smaller elements if it is believed that they were incorrectly grouped.

The KS will split an element if it is thought that the element corresponds with more than one element on the model panel. It is possible to determine that an element should be split by examining its belief function; an element that corresponds to more than one model element

will have high belief values that are nearly equal for two or more members of its FOD. For example, if two edges are configured as shown in Fig. 7.4 (a), and the nearly vertical one is believed to correspond with two model-elements because two of the members in its FOD have high belief, then the splitter KS will split it near the vertex of the other edge. In this example, the preprocessor did not form a junction between the upper and lower halves of the edge because it failed to detect the edge's intersection with another edge. On the face level, the splitter KS severs a "peninsula" from a face-element if the two edges on either side of the peninsula have the same label. This is shown in Fig. 7.4 (b).

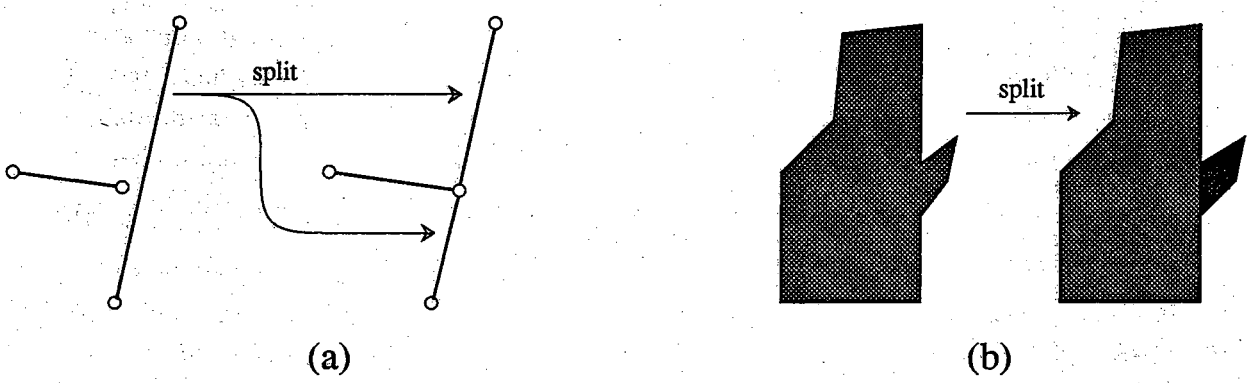


FIGURE 7.4 This figure shows the actions performed by the splitter KS. Panel (a) shows how an edge can be split to join it with another edge. Panel (b) demonstrates how a "peninsula" can be split from a face.

The splitter KS also corrects elements that were incorrectly formed by the grouper KS. For example, one or more of an element's children may not belong with the rest of the group. These elements are relatively easy to spot because the belief in their labels is usually suspiciously low when compared to the belief in their siblings' labels. Once the incorectly grouped children are discovered, it is an easy task for the splitter KS to duplicate the old parent element with the exception that the incorrectly grouped children are omitted from the duplicate's list of children. The original element is then flagged to be ignored.

It is also common for an initial grouping to be contaminated by competing children. For example, when grouping edges into a face, the grouper may include multiple renditions of the same edge in the same group. If the gray level variations corresponding to a scene edge do not exhibit a monotonic variation in directions perpendicular to the edge, the edge may be detected as multiple parallel edges in close proximity to one another. Edges 1 and 24 in Fig. 7.2 could be an example of such an artifact. An important job assigned to the splitter is the detection of such parallel edges. It does this by measuring the angle and the extent of the overlap between two grouped elements with the same label. The overlap is measured by projecting the shorter of the edges onto the longer one. When such competing parallel edges are found, multiple groupings are formed from an initial group by retaining only one competing parallel edge at a time.

In the above example, edge 24 will compete with edges 1 and 2 in F_3; the same will be the case with the edges 6 and 9 in F_4. So, the above initial groups lead to the following groups:

$$F_1 = \{E_{16}, E_{18}, E_{19}\}$$
$$F_2 = \{E_3, E_4, E_8, E_{10}, E_{11}\}$$
$$F'_3 = \{E_1, E_2, E_{14}, E_{15}, E_{20}, E_{21}, E_{23}\}$$
$$F''_3 = \{E_{14}, E_{15}, E_{20}, E_{21}, E_{23}, E_{24}\}$$
$$F'_4 = \{E_5, E_7, E_9, E_{12}\}$$
$$F''_4 = \{E_5, E_6, E_7, E_{12}\}$$

Note that the splitter KS and the merger KS do not delete elements that they believe to be incorrectly grouped; instead they create new elements and set a flag in the old element indicating that the element is no longer in focus. The old elements are not destroyed so that the KSs may check to see if a newly created element is identical to an older element that is no longer in focus. The new element is deleted immediately if it is determined to be identical to such an element. The older elements are also allowed to remain on the blackboard because, at some later time in the processing, it may be decided that they were correct and should be used.

CHAPTER 8

BLACKBOARD IMPLEMENTATION IN OPS83

Philosophically, all blackboard (BB) systems are alike in that they all contain three main components. First, they all contain a collection of knowledge sources (KSs) into which the domain knowledge is partitioned; that is, each KS is able to solve a small portion of the total task. Furthermore, blackboard systems are so named because each contains a blackboard, a hierarchical database containing the data for the specific problem on which work is being done. To keep the KSs independent, communication between them is allowed to take place only through the blackboard database. Finally, each of the systems contains a control mechanism, commonly called the scheduler, that can respond opportunistically to data residing on the blackboard in order to optimize control flow.

Although all blackboard systems are conceptually similar, implementation details affect control strategies, KS granularity, etc. This chapter will address PSEIKI's implementation in OPS83 and the effects of the rule-based programming language on design decisions. The chapter will show the working memory data structures used for representing the data-elements and the knowledge source activation records*. Subsequently, the current implementation of the scheduler and the monitor will be described. Finally, KS implementation will be described; the operation of the grouper KS will be described in detail and the operation of the labeler KS, the splitter KS and the merger KS will also be discussed.

8.1. OPS83 Data Structures Used By PSEIKI

PSEIKI uses the working memory of OPS83 for the BB data structure; each working memory element (WME) corresponding to the BB data structure describes a data-element at some level of the BB. In addition to being a host for the BB data structure, the working memory also stores the knowledge source activation records (KSARs). A KSAR is created by the BB monitor when the trigger conditions for a KS are satisfied by some data-element. (It is the job of the monitor to keep track of the data on the BB and to constantly check whether a newly created data-element satisfies the triggering conditions for a KS.) KSARs also can be created by KSs, allowing KSs to trigger other KSs explicitly. Each KSAR holds the identity of the data-element that meets the triggering conditions of a KS, the relevant KS, and other pertinent information such as the cycle during which the KSAR was created. This information

* If not already familiar with terms like "working memory," "production memory," etc. the reader is referred to [BroFar85] for a nice exposition on the architecture of a production system. The OPS83 used for PSEIKI is a direct descendent of the OPS5 system described in [BroFar85]. Much more so than OPS5, OPS83 allows functions and procedures to co-exist with rules and working memory elements.

indicates to the KS the object on which work should be performed and aids the scheduler in choosing a KSAR to activate.

8.1.1. Working Memory Elements for Representing Data

A single WME class is used to store all data-elements, regardless of the BB level at which the data-element resides. In other words, the same WME class is used for edges, faces, objects and scenes. The distinctions between different types of data-elements are introduced by using appropriate values for the **level** attribute. Using the same WME class allows generic functions to be applied to elements from all of the data levels.

Fig. 8.1 shows the definition of the WME class for representing data. Most of the WME fields are self-explanatory. The element's **id** number is a unique identifier used to keep track of individual data-elements; data-elements are always referenced via their id numbers. The **panel** and **level** fields specify the element's location on the BB. The **type** field is used to specify the type of data from which the element is derived; the values that it can assume are *two_d, three_d* and *model.*

The next two fields specify the sub-elements from which an element is built. The **children** field is used to store the list of id numbers of the element's children. The **madeof** field has a number of uses. If the element is on the data panel and was built by the splitter KS or merger KS, then this field stores the id number of the element(s) that were split or merged to form this element. However, if the element is on the model panel, then this field is used to store the id numbers of all data elements whose labels are equal to this element's id number.

The next few fields are parameters of the data-element. The **value** field is a generic attribute in which a level specific value is stored. For example, it is used to specify the strength of an edge or the average gray level of a face. The **size** parameter is also generic; this parameter is used to specify the degree, length, area or volume if an element is a vertex, edge, face, or an object, respectively. The **near** and **far** parameters are used to specify the two diagonal vertices defining the extent of the element.

The **focus** field has two functions. If the element is on the data panel, then this field is used as a flag indicating if the element is in focus; a zero value indicates that the element is no longer in focus and should not be used in further processing. If the element is on the model panel, then this field is used to specify the desired number of competing data elements that have this element as their model. For example, if the value of this field was set to three for an element on the model panel, then there should be at least three in-focus elements on the data panel that have this element as their model.

The next two parameters specify the data-element's location if it is a vertex. The **rowcol** attribute indicates a vertex's coordinate on the image plane if it was obtained from 2D data.

```
type Data=element (
    id:             integer;    -- unique id number
    panel:          integer;    -- panel in the BB
    type:           symbol;     -- type of panel (two_d, three_d, model)
    level:          symbol;     -- level in the panel (vertex, edge, ...)
    source:         symbol;     -- source of the element (original, synthetic)

    -- Parameters defining the composition of element
    children:       list;       -- children of element
    madeof:         list;       -- list of elements that were split
                                -- or merged to create this element

    -- General Parameters
    value:          integer;    -- edge strength, face grey-value, etc.
    size:           integer;    -- edge length, area of face, etc.
    near:           vector;     -- coordinate of extent
    far:            vector;     -- coordinate of extent
    focus:          integer;    -- flag set if element is in focus

    -- Parameters valid only for vertex-elements
    rowcol:         ivector;    -- (vertex) image coordinate of vertex
    coord:          vector;     -- (vertex) world coordinate of vertex

    -- Parameters used for uncertainty management
    frame:          list;       -- frame of discernment
    bpa:            bpas;       -- basic probability assignment
    positive:       real;       -- updating bpa belief
    negative:       real;       -- updating bpa disbelief
    label:          integer;    -- label of element
    belief:         real;       -- belief in label
);
```

FIGURE 8.1 This is the WME class definition for data-elements.

Likewise, the **coord** attribute specifies the vertex's location in the 3D world coordinate frame.

The remaining fields shown in Fig. 8.1 hold the uncertainty information about a data-element and are used by the labeler KS. The **frame** attribute holds the list containing the element's frame of discernment and the **bpa** attribute holds the element's basic probability assignment. An element's updating bpa is stored in the **positive** and **negative** attributes; these

values indicate the new belief and disbelief in the element's label. Finally, the element's **label** and **belief** in that label are indicated by the next two attributes.

8.1.2. The WME Class for Representing KSARs

Fig. 8.2 shows the WME class definition for representing a KSAR. The **id** field is used to keep track of the KSARs while the state of any KSAR is determined by its **status** field. The **KS** and **action** fields of the KSAR specify what action is to be performed on its focal-element. The **object** field is used to specify the id number of the KSAR's focal element; the **level** and **panel** fields specify the location of the focal element on the BB. The **using** field is used to specify the secondary focal element; for example when the merger KS is to merge two elements, the id number of the second element is stored in this field. PSEIKI's scheduler uses the **priority** field when ranking KSARs for firing; only the KSARs for the splitter KS and merger KS have non-zero priority values for reasons to be discussed later in this chapter. The **trigger_cycle**, the **trigger_KSAR** and the **active_cycle** fields are used as a log of the BB activities; they are used to record the BB cycle that a KSAR was created, the KSAR that was active when the this KSAR was created and the BB cycle on which this KSAR was run, respectively. This information has proven useful for debugging the BB.

```
type KSAR=element (
     id:              integer;    -- KSAR id #
     status:          symbol;     -- KSAR status

     KS:              symbol;     -- Knowledge source being triggered
     action:          symbol;     -- action KS is to perform

     object:          integer;    -- Object being focused on
     using:           integer;    -- Secondary object being focused on
     level:           symbol;     -- Level being focused on
     panel:           integer;    -- Panel Being focused on

     priority:        real;       -- KSAR priority.

     trigger_cycle:   integer;    -- cycle KSAR was formed
     trigger_KSAR:    integer;    -- KSAR which was active when
                                  -- this one was triggered
     active_cycle:    integer;    -- cycle during which KSAR was active
);
```

FIGURE 8.2 This is the WME class definition for KSAR.

The KSAR originally is created with its status marked as **pending**. This means that the KS has been triggered but has not yet been run. When the scheduler decides to fire on a KSAR, it marks the KSAR's status to **active**. At this point, the KS's precondition and poisoning productions are allowed to fire; it is their job to mark the KSAR's status to **running** if the preconditions are met or **poisoned** if they aren't. If the KSAR is determined to be poisoned, the KS's body productions are not allowed to fire and control is passed back to the scheduler. If the status has been set to running, the KS's body productions are allowed to fire. After the KS has accomplished its goal, it marks the KSAR's status field to **finished** and returns control to the scheduler.

8.2. Scheduler and Monitor Operation

8.2.1. Scheduler Operation

The scheduler is the heart of any BB. It is the scheduler's job to choose what action to perform at any cycle of the BB operation. It carries out this job by selecting one of the pending KSARs and activating the corresponding KS. PSEIKI's scheduler, which consists of a set of metarules, runs by default; that is, it runs automatically when no KSs are active. Initially, when data is deposited on the BB, the scheduler is invoked to get the entire process started.

PSEIKI's scheduling strategy can be broken into three phases. The first phase is called the *initialization* phase. In this phase, the labeler KS is used to assign labels to the elements deposited on the data panel by the low-level processor; the grouper KS and labeler KS are also used to create and assign labels to elements on the upper levels of the data panel, respectively. In the second phase, called the *updating* phase, the belief in the labels of the data elements are updated using the techniques presented in chapters 5 and 6. The third phase is called the *incorporation* phase; in this phase, the evidence passed up the hierarchy by the low-level elements is incorporated into the upper-level elements' belief functions.

Although scheduling algorithm follows this three phase pattern in general, the actions usually designated to one phase may be performed in another phase if the need arises. For example, during the updating phase, if all of the elements with a particular label have their label changed, then the grouper KS will be fired to try find another element that can be given the desired label. The orderly flow of BB processing may also be interrupted by scheduling the splitter KS or the merger KS because these two KSs take scheduling presidence over the grouper KS and the labeler KS. That is, the scheduler will fire the merger KS or the splitter KS as soon as one of their KSARs appears indicating that two elements should be split or merged. It seems reasonable to fire these two KSs first because it is their duty to correct misformed groups. If an element is composed of a misformed group, then any processing resources spent labeling that element or including that element in a group will most likely be

wasted. Thus, it makes sense that we try to correct these misformed groups as soon as possible.

Two actions are performed during the initialization phase of BB processing: data elements on the lower levels of the BB are grouped into elements on the upper levels by the grouper KS and the labels and belief functions of unlabeled data elements are initialized by the labeler KS. Backward chaining is used extensively to guide KSAR scheduling during the initialization phase. Scheduling is started with the goal of finding a prespecified number of competing scene elements. The number of scene elements that the scheduler tries to find is specified by the value of focus field of the only scene-level model element; the value of this field is set by the user at the start of processing. To find the competing scene elements, the scheduler creates the sub-goal of finding a prespecified number of objects in the scene in order to group them into the desired elements; once again, the number of competing object-level elements is specified by the focus field of the appropriate model element. The rule shown in Fig. 8.3 is used to chain down the expected scene creating goals and sub-goals to find elements and their children.

```
--
-- RULE    : schedule_init_children
--  IF     : We are trying to find an elements on a label that has no
--         : data-elements on it
-- THEN    : create sub-goals (contexts) to find the element's kids
--
rule schedule_init_children {
        &contxt     (Context current=sched_init_element);
        &model      (Data id=&contxt.object);
        -           (KSAR KS=label; action=initialize; level=&model.level);
        &kid  (Data in_list(@.id, &model.children));
-->
        make (Context current=sched_init_element; object=&kid.id);
};
```

FIGURE 8.3 This is the rule that chains down the model hierarchy creating goals to find the children of a model element.

This rule works as follows: The first two CEs are used to match the model element for the current goal. The third CE checks to see if there is an element on the same level as the current goal element; the rule will not fire if there is such an element. If there is data element on the current level, then the labeler KS should be fired to label it and this rule need not fire. When the rule fires, the RHS merely creates a context element (sub-goal) to find the child.

Sub-goals are created to find the elements on successively lower levels of the BB until a level is reached that contains the data elements deposited by the preprocessor. If an edge-based preprocessor was used to generate PSEIKI's input data, then the edge level will be the highest level with data elements on it; if a region-based preprocessor was used, then the face level will be the highest level with data elements on it. When a sub-goal is created to find an element on a level that contains data elements, the rule shown in Fig. 8.4 becomes enabled and fires the labeler KS to initialize the labels of the elements on this level. This rule fires once for every data element on that level of the BB.

```
--
-- RULE    : schedule_init_label
--  IF     : The is a goal to find a model element that lies on a level
--         : that contains data elements
-- THEN    : Fire the labeler KS to initialize the label
--
rule schedule_init_label {
        &contxt     (Context current=sched_init_element);
        &model      (Data id=&contxt.object);
        &ksar(KSAR level=&model.level;
                         KS=label; action=initialize; status=pending);
        ~       (KSAR priority > PRIORITY_THRESHOLD);
-->
        modify      &ksar(status=active; active_cycle=&current_cycle);
};
```

FIGURE 8.4 This rule is used to schedule the labeler KS to initialize the labels of data elements.

The LHS of this rule is very similar to the LHS of the rule in Fig. 8.3; the main difference between the two is found in the third CE. In this rule, the third CE is used to match a labeler KSAR; in the previous rule, the third CE was used to prevent the rule from firing if it matched a labeler KSAR. The last CE is used to prevent the rule from firing if there is a pending KSAR for the splitter KS or the merger KS; we will describe the scheduling algorithm used to fire these two KSs later. This rule's only action is to fire the labeler KS on the element specified by the matched KSAR. Note that only the highest level elements deposited onto the data panel are labeled at this time (e.g. faces for a region-based preprocessor); the labels for elements on the levels lower than this are not initialized until the updating phase of BB processing.

After all of the labels for these elements have been assigned, the grouper is scheduled to group them into elements on higher levels of the BB. As soon as the grouper KS forms an element, the labeler is fired to label it. The grouper is not allowed to be fired to form a new element on the data panel until each child of that element has the prespecified number of competing elements (as specified by their focus fields). The rule shown in Fig. 8.3 is used to schedule

the labeler KS and one like it is used to determine that the grouper KS should be fired. After it has been determined that the grouper should be fired, a number of rules fire that determine the child element that will be used as its seed-element. These rules rank the grouper KSARs based on the product of the element's size and the belief in its label. One of the rules used to rank the grouper KSARs is shown in Fig. 8.5.

```
--
-- RULE    : find_group_candidate
--  IF     : there is a context to find a candidate for the seed element
--         : with a particular label
-- THEN    : Choose, as the candidate, the element with the largest product of
--         : size and belief
--
rule find_group_candidate {
        &contxt     (Context current=sched_find_candidate);
        &model      (Data id=&contxt.object);
        &el    (Data label=&model.id);
        ~      (Data children[2]=&el.id);
        &ksar(KSAR object=&el.id; KS=group; action=initialize;
                        status=pending);
        [&el.belief * &el.size];
-->
        modify      &ksar(status=candidate);
        remove      &contxt;
};
```

FIGURE 8.5 This is one of the rules used to rank grouper KSARs.

This rule is used to find candidate seed elements that may be used as the seed element of the group. It finds one of these candidate elements for each of the children of the model element being formed. The first two CEs guarantee that a candidate element with a particular label is found. The third CE matches the data element that will become the candidate seed element. The fourth CE guarantees that the candidate has not been used as the seed element for another group; in effect, this prevents an element from being the seed element for more than one group. Finally, the last CE matches the grouper KSAR with the designated seed-element. The structure on the next line uses a feature of OPS83 to choose, as the candidate, the element with the largest product of size and belief. OPS83 uses the value in the square brackets to rank instantiations in the conflict set; everything else being equal, OS83 selects the rule instantiation for firing that yields the greatest value for the expression in the brackets. Thus the construct will force the rule to fire on the data element with the largest product of size and belief. When this rule fires, it flags the KSAR as a candidate and deletes the context so that the rule will not

fire again. Other rules are also used to in the grouper KSAR ranking and selection process; they are not shown here for brevity's sake.

The following scheduling scheme is used during the updating phase of BB processing. First, the labels of all of the children of the in-focus scene-level elements are updated* (all of which will reside on the object-level). Next, the labels of all of the children of the object-level children are then updated (all of these elements will reside on the face-level). This updating process proceeds down the data-panel hierarchy in a depth-first manner until the edge level is reached. If an edge-based preprocessor was used to provide the input data, then labels will have been assigned to the edges during the initialization phase; in this case, the edges' belief functions are updated normally. However, if a region-based preprocessor was used to provide the input data, then the edges on the data panel will not have been labeled during the initialization phase. In this case, labels are assigned to the edges and then the belief in these labels is updated.

The following rule (Fig. 8.6) fires the labeler KS to update the belief in an element's childrens' labels. The first two CEs in the LHS of this rule are used to match the KSAR to fire. The last CE of this rule is used to prevent the rule from firing if there is a pending splitter KSAR or merger KSAR. The RHS of this rule changes the status of the labeler KSAR to active, causing the KS to fire.

```
--
-- RULE    : fire_on_update_element
--  IF     : There is a context to update the children of an object
--         : AND there is also a KSAR to update the children
-- THEN    : fire the KSAR
--
rule fire_on_update_element {
      &contxt     (Context current=sched_update_element);
      &ksar(KSAR object=&contxt.object;
                  KS=label; action=update; status=pending);
      ~     (KSAR priority > PRIORITY_THRESHOLD; status=pending);
-->
      modify      &ksar(status=active);
};
```

FIGURE 8.6 This rule is used to fire the labeler KS during the updating phase of BB processing.

* Note: the labeler KS does not update the belief in the focus element's label; rather, it updates the belief in the labels of the focus element's *children*.

The following rule (Fig. 8.7) will fire after an element's childrens' labels have been updated; this rule generates a context to force the scheduler to fire the labeler KS on the element's grandchildren. The first CE of this rule allows the rule to fire only if the BB is in the updating phase of processing. The second CE makes sure that the labeler has fired on an element before it fires on the element's children. The third and forth CEs match the element and one of its children, respectively. The last CE makes sure that the rule fires only once with any element/child pair. When the rule fires, it creates a context to fire the labeler KS on the child found by the forth CE. The context WME matched by its first CE is then modified so that it will be the most recent WME in the working memory. Making the context the most recent WME forces the rule to generate the update context for all possible element/child pairs before any other rule is allowed to fire.

```
--
-- RULE    : schedule_update_children
--  IF     : An element's children have been updated
-- THEN    : Generate contexts to update the children of the children
--
rule schedule_update_children {
        &contxt     (Context current=sched_update_element);
        &ksar(KSAR object=&contxt.object;
                    KS=label; action=update; status<>pending);
        &el   (Data id=&contxt.object);
        &kid  (Data in_list(@.id, &el.children));
        ~     (Context current=sched_update_element; object=&kid.id);
-->
        --
        -- Make the context to schedule the child
        -- also modify the current context last so that this rule
        -- will fire next (because of recency) to find any other kids.
        --
        make (Context current=sched_update_element; object=&kid.id);
        modify      &contxt     ();
};
```

FIGURE 8.7 This rule makes the scheduler fire the labeler KS to update the labels of elements on the lower levels of the BB.

The incorporation phase of BB processing is used to accumulate, into an element's belief function, the evidence that was generated by checking the consistency of the element's descendents. The scheduling scheme used in this phase of processing is identical to that used in the updating phase. That is, the labeler KS is fired first on the in-focus scene elements, and then

their children, followed by their children's children, etc. The similarity of the two scheduling schemes is reflected in the similarity of rules implementing the strategies; in fact, the rule used in the incorportation phase is identical to the rule shown in Fig. 8.6 except that the context is called "sched_incorp_element" instead of "sched_update_element" and the KSAR's action is now "incorp_update." Because, the rules used to perform the scheduling in this phase of BB processing are so similar to the rules used in the updating phase, they will not be shown here.

Scheduling the firing of the splitter KS and the merger KS is viewed as an exceptional event which is not part of the normal KSAR selection process. These two KSs are viewed in the exceptional manner because they are used to correct misformed groups; thus, they would not be needed if the low-level preprocessors always produced error-free results and the grouper KS always produced correct groupings. Because the splitter and merger are used in this manner, PSEIKI's overall scheduling scheme can best be thought of as the label/update/incorporate process described above with opportunistic interruptions made by the splitter and merger to correct misformed groups. We will now describe the process used by the scheduler to interrupt the normal label/update/incorporate flow of control when the grouper or splitter needs to be fired. The orderly flow of BB processing is interrupted as soon as a high priority splitter KSAR or merger KSAR appears in the working memory and does not resume until all of these exceptional KSARs have been fired upon. When one or more splitter KSARs or merger KSARs appears in the working memory, the scheduler ranks them based on the value of their *priority* field and chooses the highest ranked KSAR for firing. The priority field is used specify the degree to which it is believed that the elements need to split or merged. If two or more KSARs have the same maximum priority value, then one is selected at random. The scheduler will continue to fire these KSARs until there are no pending splitter KSARs or merger KSARs left in working memory that have a priority greater than a predefined threshold. Note that the priority field is always zero for labeler KSARs and grouper KSARs because they are scheduled using the scheme described above. The rule shown in Fig. 8.8 selects the splitter or merger KSAR with highest priority and fires on it.

8.2.2. Monitor Operation

The monitor is the watchdog of the BB. It is the monitor's job to keep track of the data on the BB and trigger the KSs when specific conditions are met. It also is up to the BB monitor to watch the BB and determine if the status of any poisoned KSARs should be reset to pending. This resetting of a KSAR's status occurs if the KS action on the specified data-element once again becomes valid. It also is up to the monitor to determine if any poisoned KSARs should be deleted; deletion occurs if there is no chance that the KSAR could once again become valid.

The BB monitor makes extensive use of OPS83 demons. A demon in OPS83 is a rule whose first CE is not a context, goal or KSAR. Because of the OPS83 rule selection strategy,

```
--
-- RULE    : schedule_fire_interrupt
--  IF     : If there is splitter or merger KSAR with priority greater than
--         : the priority threshold and there is no KSAR with higher priority
-- THEN    : Fire on the KSAR
--
rule schedule_fire_interrupt {
        &ksar(KSAR priority > PRIORITY_THRESHOLD);
        ~       (KSAR priority > &ksar.priority);
-->
        modify      &ksar(status=active);
};
```

FIGURE 8.8 This rule is used to fire the splitter KS or merger KS.

these rules take precedence over ordinary rules (e.g. rules inside of KSs or scheduler rules) and fire as soon as they become completely instantiated. Thus a demon in OPS83 can be thought to operate outside of any context, KS or goal search.

As an example of a monitor rule, consider Fig. 8.9. This rule, used to trigger the grouper KS, fires when it finds a data-element without any parents (an orphan element). The rule then creates a KSAR that directs the grouper KS to find the element's parents. This rule works as follows: The first CE matchs any new data-element if it has a label; this data-element is the *focus-element* of the rule. The second CE allows the rule to fire only if the focus-element is an orphan. This CE uses the function *in_list(·)* to match any WME that has the first CE's id number in its list of children. The tilde in front of the CE acts as a negation symbol; that is, it allows the rule to fire only if no WME matches the CE. Thus the tilde in front of the second CE of this rule keeps the rule from firing if the focus-element has a parent. The last CE keeps the rule from firing if the grouper KS already has been triggered on this data-element; the rule is fired if a pending grouper KSAR focused on the same element can not be found.

8.3. Operation of the KSs

Even though the various KSs perform very different tasks, many common subtasks are performed by all of them during KS operation. These subtasks start when the scheduler marks a KSAR's status to active. After a KS becomes active, its **poisoning** rules are allowed to fire; these rules make sure that the KS's preconditions have not become invalid since the KS was triggered. If a poisoning rule does fire, it sets the KSAR's status to poisoned and returns control to the scheduler. If none of the poisoning rules fire, a rule that marks the KSAR's status to running fires by default.

```
--
-- RULE    : group_trigger
--  IF     : There is a labeled element that is being focused on but
--         : has not yet been placed in a group
--         : AND a KSAR saying that it should be grouped has not yet been created
-- THEN    : Create a KSAR that indicates that the element should be grouped
--
rule group_trigger {
      &el   (Data type<>model; label<>0; focus<>0);
      ~     (Data in_list(&el.id, @.children));
      ~     (KSAR KS=group; action=initialize; object=&el.id);
-->
      make (KSAR KS=group; action=initialize;
            trigger_cycle=&current_cycle;
            id=&next_KSAR_id; status=pending;
            object=&el.id; panel=&el.panel; level=&el.level;
            priority=0.5);
      &next_KSAR_id = &next_KSAR_id + 1;
};
```

FIGURE 8.9 This is a monitor demon that is used to create a KSAR for the grouper KS.

After the KS starts running, the control flow becomes more KS specific, but it still follows the same pattern. The first few rules that fire after the KS starts running usually are *driver rules*. These rules don't contribute directly to the solution of the KS's task; instead, they initialize, in working memory, the elements that the KS needs to solve the task. These driver rules can generate contexts needed by the KS in its problem solving activity. They also can put on the BB dummy data-elements that will be "fleshed out" during the course of the KS's processing. After the KS's driver rules are fired, the control flow becomes very KS specific. In the next few sections, the control flow inside each KS will be demonstrated through the use of a few examples.

8.3.1. Grouper KS Operation

To illustrate the flow of control inside a KS, the grouper KS's formation of a face from edges will be examined. The example in Fig. 8.10 will be used to make the explanation more concrete. Assume for the example that the grouper KS has been activated with a KSAR focused on the element E_9 of Fig. 8.10. As previously described, the KS's poisoning rules are allowed to fire when it is first activated. Fig. 8.11 is an example of a poisoning rule used by the grouper KS. This rule is meant to poison a KSAR if the grouper KS fires on an element

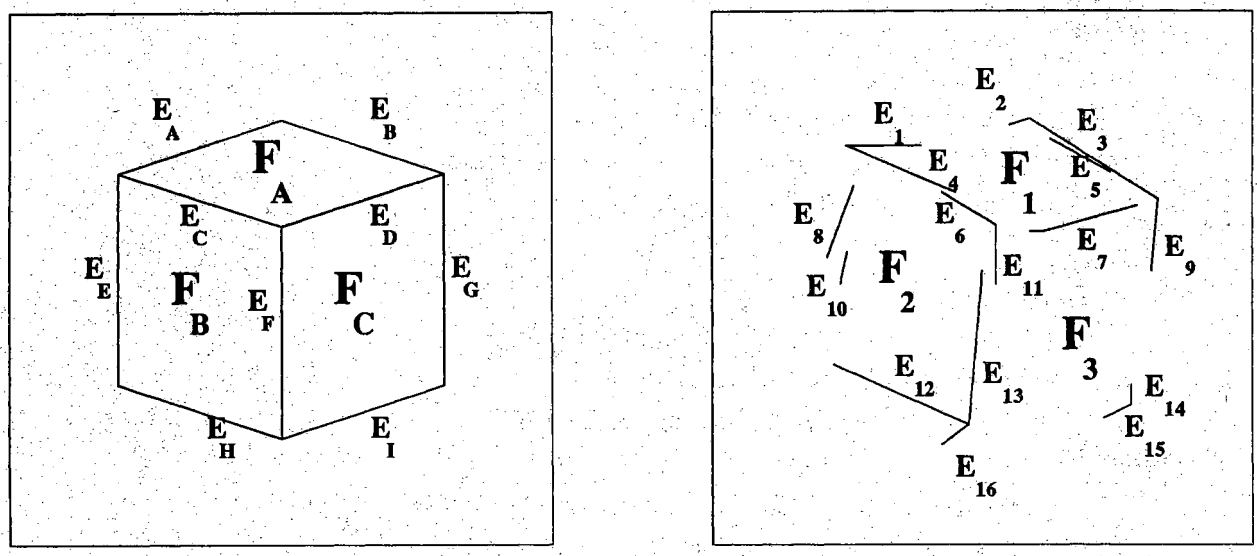


FIGURE 8.10 This figure shows an example of data on the BB and is used to explain KS operation.

```
--
-- RULE    : edge_group_poison
--  IF     : The active KSAR focuses on an edge that is already the seed of a group
-- THEN    : Poison the KSAR
--
rule edge_group_poison {
        &ksar(KSAR KS=group; action=initialize; status=active);
        ~       (Data level=face; children[2]=&ksar.object);
-->
        modify      &ksar(status=poisoned);
};
```

FIGURE 8.11 An example of a poisoning rule.

has already been used as the seed of a group*. This rule works in the following manner: The first CE matches the active KSAR if its action is to initialize a group. The second element determines if there is a face-element that has the focus element as its seed. The seed element of a group is always stored in the second position the element's list of children (the position of

* This does not imply that a data-element can participate only in a single group. An edge-element, for example, is allowed in two or more groups if it is on the common boundary between them. However, an edge-element can serve as a seed for only one group. Therefore, an edge-element that belongs to two or more groups can trigger the formation of only one of them; other edges would have to act as seeds for the other groups.

the last element in a list is stored in its first position). If this CE matches a WME, then the focus-element already was used as a seed; the rule fires, and the KSAR is marked as poisoned. If no poisoning rules fire, another rule fires by default and marks the KSAR's status to running. Thus if it is assumed that element E_9 has not been used as the seed for another group, then the active KSAR's status is set to running.

The grouper KS uses a driver rule to initialize internal processing; this rules fire immediately after the KS starts running. The driver rule is used to deposit an element on the BB that will be used as the focus element's parent. The grouper KS then finds other elements on the BB that can become siblings of the focus element and groups them into the focus element's parent. Fig. 8.12 shows the driver rule for group initialization.

The rule in Fig. 8.12 works as follows: The first two CEs match the running KSAR and the focus-element. The third CE prevents the rule from firing if it detects the focus-element's parent. Because this rule creates the focus element's parent, the third CE prevents the rule from firing more than once during any KS activation. The last CE is designed to find a possible model for the parent-element by finding the parent of the focus-element's label-element.

The rule performs two actions when it fires. First, it builds the parent-element. As mentioned previously, the KS's focus-element and its siblings will be grouped into this element. The parent-element is initialized with appropriate parameters: panel, data type, level, id number, size, etc. The parent-element's seed element is set to the focus-element. This is done to prevent the driver rule from firing twice and to allow the remaining KS body rules to find both the focus and parent-elements easily. The rule also builds a KSAR that requests that the parent-element be labeled.

Because edge E_9 is an orphan in the example, this driver rule would fire. When the rule fires, a new element, say element F_3, is created and depositied on the BB. This new element lies on the face level of the data panel with label F_C and, initially, has element E_9 as its only child. Now it is up to the rest of the KS body rules to find element E_9's siblings and group them into face F_3.

After the driver rule initializes the parent-element, the remaining KS body rules can fire. Only one KS body rule needs to fire to group edge-elements into the face-element. This rule (shown in Fig. 8.13) fires at least once for every edge that can be grouped into the face.

The first four CEs of the rule in Fig. 8.13 find the active KSAR, the parent-element, the model of the parent-element and the focus-element, respectively. The fifth CE finds a candidate to group into the parent. This CE makes sure that the candidate is on the same level and panel as the focus-element and that it has not yet been grouped into the parent. Furthermore, this CE makes sure that the label of the candidate allows it to be grouped into the parent by checking to see if the candidate's label element is a child of the parent's label element. The rest of the CEs merely obtain data needed in the right hand side (RHS) of the rule. The sixth

```
--
-- RULE    : group_driver -- start up grouping process by creating parent element
--         :          -- also make KSAR to label parent element
--  IF     : There is a running KSAR that says to initialize a group
--         : AND there is no element with the KSAR's object as its main child
-- THEN    : Make the parent element and a KSAR to label it
--
rule group_driver {
      &ksar(KSAR KS=group; action=initialize; status=running);
      &el   (Data id=&ksar.object);
      ~     (Data children[2]=&ksar.object);
      &model     (Data in_list(&el.label, @.children));
-->
      &max_id = &max_id + 1;
      make (Data id=&max_id; source=synthetic; label=&model.id;
                  type=&el.type; panel=&el.panel; level=&model.level;
                  size=&el.size; value=&el.value; focus=1;
                  near[1]=&el.near[1]; near[2]=&el.near[2];
                  near[3]=&el.near[3]; near[4]=1.0;
                  far[1]=&el.far[1]; far[2]=&el.far[2];
                  far[3]=&el.far[3]; far[4]=1.0;
                  children[1]=2; children[2]=&el.id);
      write () |initializing |, &model.level, | |, &max_id, '0;
      write () |   grouping |, &el.level,    | |, &el.id, '0;
      make (KSAR KS=label; action=initialize;
                  trigger_cycle=&current_cycle;
                  id=&next_KSAR_id; status=pending;
                  level=&model.level; object=&max_id; priority=0.5);
      &next_KSAR_id = &next_KSAR_id + 1;
};
```

FIGURE 8.12 An example of a driver rule.

CE, one of the CEs used to obtain data for the RHS, matches a WME that holds a homogeneous transformation matrix. The transformation matrix is defined to transform the focus-element's label-element so that it is compatible with the candidate's label-element.

When the rule fires, the compatibility between the candidate and a transformed version of the focus element is computed as described in chapter 5. If this value is greater than a threshold, then the *add_list()* function is used to add the candidate's id number to the parent's list of children. Notice that if the candidate-element doesn't meet the criteria to be grouped, then

```
--
-- RULE : group_into_face -- group edge-elements into a face-element
--  IF    : We are grouping edges into a face and there is a compatible edge
--        : that is not yet in the face
-- THEN : IF the xformed version of the edge is collinear with the focus
--        : element, put it into the group
--
rule group_into_face {
        &ksar    (KSAR  KS=group; action=initialize; status=running);
        &face    (Data children[2]=&ksar.object);
        &model  (Data id=&face.label; level=face);
        &edge1  (Data id=&ksar.object);
        &edge2  (Data type<>model; level=edge; id<>&edge1.id;
                        in_list(@.label, &model.children);
                        (~in_list(@.id, &face.children)));

        -- get parameters needed in rhs computations
        &xfrm    (Model_xfrm from=&edge2.label; to=&edge1.label);
        &s1      (Data id=&edge1.children[1]);
        &e1      (Data id=&edge1.children[2]);
        &s2      (Data id=&edge2.children[1]);
        &e2      (Data id=&edge2.children[2]);

        &slop    (Constant name=max_dist);
        &dist    (Constant name=group_threshold);
-->
        local &compat, &incompat:  real;
        local &belief, &disbelief: real;

        call edge_compatibility(&s1.coord, &e1.coord, &s2.coord, &e2.coord,
                                &xfrm.xfrms, &slop.real_value,
                                &compat, &incompat);
        &belief    = &compat   * &edge2.belief * &xfrm.scale_fact;
        &disbelief = &incompat * &edge2.belief * &xfrm.scale_fact;

        if (&compat > &dist.real_value) {
                modify &face  (call add_list(&edge2.id, @.children);
                        size = @.size + &edge2.size;
                        call update_belief(@, LEVEL_SCALE * &belief,
                                                LEVEL_SCALE * &disbelief));
                write ()|   grouping edge |, &edge2.id, '0;
        };
};
```

FIGURE 8.13 This rule is used to group edges into faces.

nothing in the working memory is changed and refraction prevents the rule from firing again with the same instantiation.

In the example, any edge that has one of the labels E_D, E_F, E_G or E_I is a candidate to be grouped with edge E_9 into face F_3. Edges E_7, E_{11}, E_{13}, E_{14} and E_{15} meet this criterion. Thus any of these edges that was compatible with the transformed version of the focus-element

would be grouped into the parent. If all but E_{11} were compatible with the transformed E_9 then the children of F_3 would be edges E_7, E_9, E_{13}, E_{14} and E_{15}.

8.3.2. Labeler KS Operation

The labeler KS can perform three actions: It can initialize a data element's label, update the belief in the labels of an element's children and incorporate, into an element's belief function, the updating belief generated by the element's descendants. These three actions are specified by setting the **action** field of a KSAR to *initialize*, *update* and *incorporate*, respectively. In this section, we will use the previous example to demonstrate the processing performed by the labeler KS to update the belief in the labels of an element's children. Using this example, we will show how the belief in the labels of the children of face F_3 are updated.

If we assume that no poisoning rules fire after scheduler activates the labeler KS, then the driver rule shown in Fig. 8.14 fires and generates context elements specifying that every child of the focus element should be used to update the belief in the label of every other child. This rule is easily understood. On the LHS, the first two CEs are used to match the KS's focus element and the third CE is used to keep the rule from firing more than once for any KS invocation. The RHS of the rule contains a set of nested *for* loops that index through the focus element's list of children and generate the desired contexts. After the driver rule has fired, the contexts are used to specify the elements that can be used to update the belief in other elements. In the example, two contexts would be generated for each pair of elements in $\{E_7, E_9, E_{13}, E_{14}, E_{15}\}$ specifying that each element should be used to update the belief in the label of every other element.

The rule shown in Fig. 8.15 is used to generate the updating evidence for edge elements. This rule fires once for every context generated by the previous rule if the belief in the edge providing the evidence is above a user specified threshold. If the belief in the edge is below the threshold, then the context is removed automatically. The first two CEs of this rule match the edge whose belief is being updated. The third CE matches the edge providing the updating evidence and also keeps the rule from firing if the belief in that edge's label is below a threshold. The remaining CEs are used to obtain the data needed by the RHS of the rule. The fourth CE is used to obtain the homogeneous transformation matrix needed to determine the compatibility of the two edges. Finally, the last four CEs are used to obtain the endpoints of the two edges. The RHS of this rule uses the *edge_compatibility()* function to measure the compatibility of the transformed version of the first edge with the second edge based on their collinearity. After the compatibility is measured, the *update_bpa()* function is used to accumulate the new evidence into the updating bpa. Finally, the context WME is removed to prevent the rule from firing again with the same context.

```
--
-- RULE    : update_certainty_driver
--  IF     : The labeler KS was just activated
-- THEN    : Generate a context to update the label of every child using
--         : the label of every other child.
--
rule update_certainty_driver {
        &ksar(KSAR KS=label; action=update; status=running);
        &el   (Data id=&ksar.object);
        ~     (Context current=incorporate_belief);
-->
        local &i, &j, &kids: integer;

        write () |updating children of |, &ksar.level, | |, &ksar.object, '0;
        make (Context current=incorporate_belief; object=&ksar.object);

        &kids = &el.children[1];
        for &i = (2 to &kids)
                for &j = (2 to &kids)
                        if (&i <> &j)
                                make (Context current=update_certainty;
                                        object=&el.children[&i];
                                        using =&el.children[&j]);
};
```

FIGURE 8.14 This is the driver rule for the labeler KS.

8.3.3. Splitter KS and the Merger KS Operation

We will illustrate the flow of control inside the splitter KS by examining the rules used to split a face with competing edges into multiple faces with one competing edge apiece. The splitter KS uses a driver rule to initialize processing; this rule is used to generate a context that directs the KS to examine the focus element for competing edges. After the driver rule fires, a level-specific body rule is allowed to fire that finds all the competing children of an element; this rule fires at least once for every pair children that could possibly compete. For example, the splitter KS uses the rule shown in Fig. 8.16 to find competing edges that the grouper has included in a face. When it finds a pair of competing edges, it creates two new faces each with only one of the competing edges; it also resets the focus flag in the original face to prevent its use in further BB processing. The rule works as follows: The first two CEs are used to match the newly created face element; they also keep the rule from firing more than once. The second two CEs are used to match two edges from the face's list of children if they have

```
--
-- RULE    : update_edge_certainty
--  IF     : there is a context to use one edge to update the belief in another's label
--         : AND the belief in the one providing the evidence is > BELIEF_THRESH
-- THEN    : update certainty based on the (in)compatibility of the two edges
--
rule update_edge_certainty {
        &contxt     (Context current=update_certainty);
        &el1  (Data id=&contxt.object; level=edge);
        &el2  (Data id=&contxt.using; belief > BELIEF_THRESH);
        -- get parameters needed in rhs computations
        &model      (Model_xfrm from=&el1.label; to=&el2.label);
        &s1   (Data id=&el1.children[2]);
        &e1   (Data id=&el1.children[3]);
        &s2   (Data id=&el2.children[2]);
        &e2   (Data id=&el2.children[3]);
-->
        local &s1_xfrm, &e1_xfrm: vector;
        local &pos, &neg: real;

        call edge_compatibility(&s2.coord, &e2.coord,
                                &s1.coord, &e1.coord,
                                &model.xfrms, distance(&s2.coord, &e2.coord),
                                &pos, &neg);
        &pos = &pos * &el2.belief * &model.scale_fact;
        &neg = &neg * &el2.belief * &model.scale_fact
                * scale_certainty(&el2);
        write () | new evidence for |, &el1.level, | |, &el1.id;
        write () | using |, &el2.level, | |, &el2.id, '0;
        modify &el1         (call update_belief(@, &pos, &neg));
        remove &contxt;
};
```

FIGURE 8.15 This rule is used to update the belief in an edge's label.

identical labels; these edges could possible compete. When the rule fires, the function *edge_overlap()* is used to determine the overlap of the two edges using the technique described in chapter 7. If the overlap is found to be greater than a preset threshold, then the two edges are considered to be competing and the face is split into two faces with one competing edge apiece. Finally the rule generates a context to check each of the new edges for other

```
--
-- RULE    : face_split
--  IF     : There is a face with two edges that have the same label
-- THEN    : Check to see if they are competing.  If so, create two
--         : faces with one competing edge apiece. Also reset the original
--         : face's focus flag.
--
rule face_split {
        &contxt     (Context current=find_competing);
        &face(Data id=&contxt.object; focus<>0);
        &edge1      (Data in_list(@.id, &face.children));
        &edge2      (Data label=&edge1.label; in_list(@.id, &face.children);
                    id<>&edge1.id; length<&edge1.length);
-->
        if (edge_overlap(&edge1, &edge2) > OVERLAP_THRESHOLD) {
                -- remove the face from the focus set
                modify &face (focus = 0);
                -- initialize the first face
                &max_id = &max_id + 1;
                make (Data duplicate_Data(&face);
                        &id = &max_id;
                        call delete_list(@.children, &edge1.id);
                        madeof[1]=2; madeof[2]=&face.id);
                make (Context current=find_competing; object=&max_id);
                &max_id = &max_id + 1;
                -- initialize the second face
                make (Data duplicate_Data(&face);
                        &id = &max_id;
                        call delete_list(@.children, &edge2.id);
                        madeof[1]=2; madeof[2]=&face.id);
                make (Context current=find_competing; object=&max_id);
        };
};
```

FIGURE 8.16 This rule is used to split a face with competing edges into multiple faces with one of the competing edges apiece.

competing edges.

The merger KS does not require any driver rules. When the KS is activated by the scheduler, a level-specific rule is fired to merge the KS's focus element with the secondary focus element. For example, the rule shown in Fig. 8.17 is used to merge two faces. The LHS side of this rule is used to match the two focus elements and guarantee that the rule will fire only once. The RHS of the rule builds an element on the face level of the data panel into which the two focus elements are merged. The children of this new element is set to be the exclusive-or of the children list of the two focus elements. The focus flags of the original two faces are also reset to prevent their use in further BB processing.

```
--
-- RULE    : face_merge
--  IF     : The active KSAR indicates that two faces should be merged
-- THEN    : merge them
--
rule face_merge {
        &ksar(KSAR KS=merge; status=running; level=face);
        &el1  (Data id=&ksar.object; focus<>0);
        &el2  (Data id=&ksar.using;  focus<>0);
-->
        &max_id = &max_id + 1;
        make (Data id=&max_id; source=synthetic;
                    type=&el1.type; panel=&el1.panel; level=&el1.level;
                    size=&el1.size+&el2.size;
                    value =  weighted_average(&el1.value, &el1.size,
                                              &el2.value, &el2.size);
                    label=&el1.label; focus=1;
                    call near_vert(&el1.near, &el2.near, @.near);
                    call  far_vert(&el1.far,  &el2.far,  @.far);
                    call xor_list(&el1.children, &el2.children, @.children);
                    madeof[1]=3; madeof[2]=&el1.id; madeof[3]=&el2.id);
        modify      &el1  (focus=0);
        modify      &el2  (focus=0);
        write () |merging into |, &el1.level, | |, &max_id, '0;
        write () |   merging  |, &el1.level, | |, &el1.id, '0;
        write () |       and  |, &el2.level, | |, &el2.id, '0;
};
```

FIGURE 8.17 This rule is used to merge two faces into a larger face.

CHAPTER 9

COMPLEXITY ISSUES IN BLACKBOARD PROCESSING

In the most general sense, PSEIKI's geometric matching activity can be expressed as the problem of finding subgraph-isomorphisms, a known NP-complete problem [GarJoh79]. It is well known that artificial intelligence's use of hueristics can greatly improve the computational efficiency of the solution to a problem solving task; in fact, it has been shown that some heuristics can beat the exponential explosion associated with NP-complete problems [Pea84]. It is hoped that the heuristics encoded into the PSEIKI's opportunistic control flow and geometric constraints, when combined with the hierarchical structure of the matching task, will enable PSEIKI to perform matching as scene complexity grows.

There are a number of ways that a system's time and space complexity can be analyzed. If the system's solution to a task can be expressed in a simple, algorithmic fashion, then its complexity often can be calculated theoretically [AhoHop74]. If a system's solution can not be expressed in a way that allows its complexity to be analyzed directly, then the system's major components can be modeled and the model analyzed. Petri net theory [Pet81], one technique for modeling systems, will be explored in this chapter. Particular attention will be focused on stochastic Petri nets, an extension to Petri net theory created by associating an exponentially distributed firing time with each transition in the net [Mol82]. Stochastic Petri nets can be analyzed by mapping the state-space of the net to a Markov-chain and by using concepts from queuing-theory to analyze the system. Currently, stochastic Petri nets can model only small-scale systems because the state-space of a Petri net grows exponentially with the size of the net (hence, so do the nodes in the Markov-chain).

If a system is too complex to be analyzed theoretically or modeled effectively, as is currently the case with blackboard systems, the only resort is to determine empirically the system's computational complexity. In the past, experimental investigations have been used to study how control flow [GarCor87] and data locking [FenLes77] affect blackboard performance. Note, since PSEIKI's hierarchical structure and geometric constraints have been fixed, PSEIKI's computational efficiency can be increased mainly by optimizing its control flow. At this time, PSEIKI's scheduler is evolving too rapidly to justify an empirical performance analysis. However, as PSEIKI's scheduler becomes more stable, an empirical performance study will be undertaken to determine how PSEIKI's matching scheme scales with problem size.

9.1. System Modeling with Petri Nets

Petri Net theory is a graph based modeling technique that has proven very powerful for modeling concurrent, synchronous and asynchronous systems. Since their introduction by C.

A. Petri in his Ph.D. dissertation [Pet66], Petri nets have been used to model complex systems in many diverse domains, some of these domains include the modeling of production systems, chemical reactions and legal systems (see [Pet81] for a bibliography of some domains of application). Because Petri nets have been used to model such a wide variety of systems and have been used by researchers with a wide range of backgrounds, they have been formulated in many different ways. The definition and development of Petri nets in this report will follow that found in [Pet81]; the reader is referred there for a more complete introduction to Petri net theory and some typical applications.

Formally, a *Petri net graph* is a directed, bipartite multigraph, $G = (V, A)$. V is the set of vertices, $V = \{v_1, v_2, \cdots, v_s\}$ and A is the set of arcs, $A = \{a_1, a_2, \cdots, a_r\}$ where an arc, a_i from vertex v_j to vertex v_k is expressed as $a_i = (v_j, v_k)$ with $v_j, v_k \in V$. Since the graph is bipartite, the set of vertices, V, can be partitioned into two disjoint parts, $P = \{p_1, p_2, \cdots, p_m\}$ and $T = \{t_1, t_2, \cdots, t_n\}$ such that each arc in A contains exactly one vertex in P and one vertex in T. Using the normal terminology, the set P is called the set of *places* and the set T is called the set of *transitions*.

A *Petri net structure*, C, is a four-tuple $C = (P, T, I, O)$. P and T are places and transitions as described previously. The input and output functions, I and O, respectively, map transitions, t_j, to collections of places. The collection of places $I(t_j)$ and $O(t_j)$ are called the input and output places for transition t_j. The *multiplicity* of the arcs between a transition and one of its input places is equal to the number of arcs from the place to the transition. Likewise, the multiplicity of the arcs between a transition and one of its output places is equal to the number of arcs from the transition to the place*. The *marking* of a Petri net is a mapping, μ, from the set of places to the non-negative integers, N.

$$\mu: P \rightarrow N$$

$\mu(\cdot)$ defines the *state* of the net. During execution of the Petri net, the marking of the net may change; that is, the function $\mu(\cdot)$ may change reflecting the evolving state of the net. The formal definition of a *marked Petri net structure* (hereafter merely called a Petri net) is $M = (P, T, I, O, \mu)$ where the components previously have been defined.

Although Petri nets are defined in abstract, graph-theoretic terms, it is often helpful to draw the marked Petri net graph. When drawing Petri nets, a *bar* | represents a transition and a *circle* O represents a place. Tokens, drawn as small dots • in a given place, p_j, are used to represent the value of $\mu(p_j)$. An input place of a transition is indicated by an *arrow* from the place to the transition. Conversely, an output place of a transition is indicated by an arrow

* Note that the input and output multiplicities between a transition and a place need not be equal if the place is both an input place and an output place for the transition. The multiplicities will differ if the number of arcs from the place to the transition is different from the number of arcs from the transition to the place.

from the transition to the place. Fig. 9.1 shows an example of a simple Petri net; Fig. 9.2 shows its associated graph.

$$P = \{p_1, p_2, p_3, p_4, p_5\}$$

$$T = \{t_1, t_2, t_3, t_4\}$$

$$I(t_1) = \{p_1\} \qquad O(t_1) = \{p_2, p_3, p_4, p_4\}$$

$$I(t_2) = \{p_2, p_3, p_4\} \qquad O(t_2) = \{p_2\}$$

$$I(t_3) = \{p_4, p_4\} \qquad O(t_3) = \{p_5\}$$

$$I(t_4) = \{p_5\} \qquad O(t_4) = \{p_3, p_4\}$$

$$\mu(p_1) = 1; \quad \mu(p_2) = 0; \quad \mu(p_3) = 0; \quad \mu(p_4) = 2; \quad \mu(p_5) = 1$$

FIGURE 9.1 This figure shows an example of a simple Petri net.

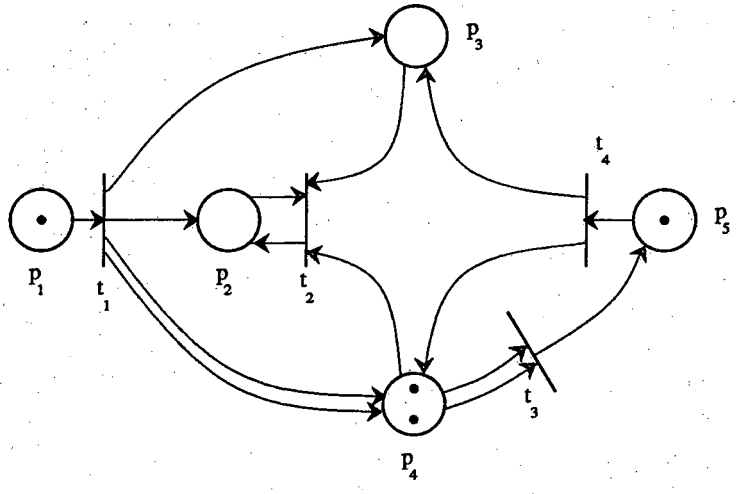


FIGURE 9.2 This figure shows the marked Petri net graph for the Petri net given in Fig. 9.1.

A transition is said to be *enabled* when the number of tokens in each of the transition's input places is greater than or equal to the multiplicity of the arcs between the transition and that input place. For example, if there are two arcs from an input place to a transition, then the transition will not be enabled until there are at least two tokens in that input place. An enabled transition is *fired* by removing tokens from the transition's input places and adding tokens to the transition's output places. The number of tokens removed from or added to the transition's input places or output places, respectively, is equal to the multiplicity of the arcs between the transition and the places. If more than one transition is enabled at any time, then the transition that is fired is picked at random. In general, the state of the net will change when a transition fires. Thus some transitions that were previously enabled may no longer be enabled and some new transitions may become enabled. The process of successively firing enabled transitions is

called *executing* the Petri net. When there are no enabled transitions, the execution of the Petri net *halts*. Fig. 9.3 shows the execution of the Petri net shown in Fig. 9.1. Panel (a) in this figure shows the net's initial marking. Panel (b) shows the net's marking after t_4 fires and panel (c) shows the net's marking after t_1 fires.

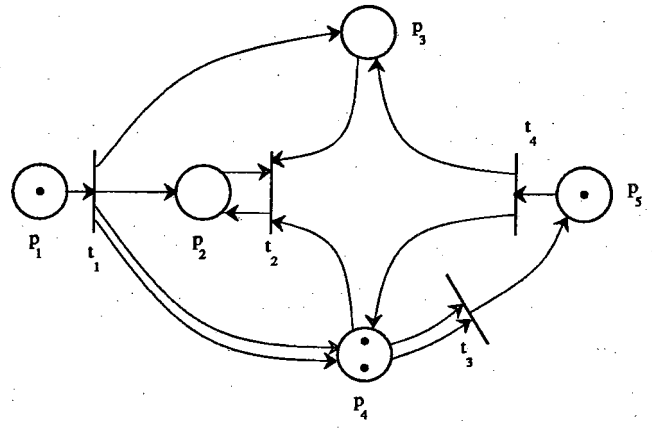
A marking of a Petri net is said to be *reachable* from another marking if there is a sequence of transition firings that transforms the state of the net from the initial marking to the desired marking. The *reachability set* of a marking is defined to be the set of all states reachable from the initial marking. Note that the reachability set of a Petri net is dependent on the original marking. Also note that the reachability set of a Petri net will grow exponentially with the number of places, transitions, and tokens present in the net. Both of these effects limit the usefulness of Petri nets in the modeling of blackboard systems.

Fig. 9.4 is a simple example of a Petri net that could be used to model PSEIKI's flow of control. The places in this net correspond with the blackboard scheduler and knowledge sources. The token represents the locus of processing in the system; a process is considered active when its corresponding place contains the token. Notice that the configuration of the net forces the control of the system to return to the scheduler between each knowledge source activation. The net can be extended to model concurrent blackboards by adding a token for each processing thread. Obviously, the model shown here is over-simplified and cannot be used in any realistic analysis.

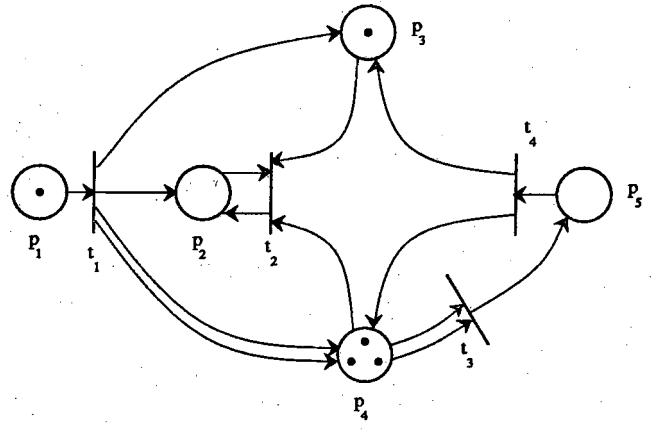
Petri net theory has been extended in a number of ways to make it a more powerful modeling tool. Stochastic Petri Nets, an extension first proposed by Molloy [Mol82], are created by associating an exponentially distributed firing time with each transition. The firing time of a transition specifies the average amount of time that the transition takes to fire. Thus the transitions in a stochastic Petri net will fire a random amount of time after they become enabled (unless another transition fires first and disables the first transition). If another transition fires but does not disable the first transition, then the timing of the first transition does not change (the first transition does not have to be "reset" because of the memoryless property of the exponential distribution).

A stochastic Petri net is formally defined as $S = (P, T, I, O, \mu, \lambda)$ where λ is the mapping from the transitions to the real numbers that defines the mean firing time of the exponentially distributed random processes. The rest of the components of S have been defined previously. Note that the transitions' firing rates are completely specified by λ because an exponential distribution is completely specified by its mean value.

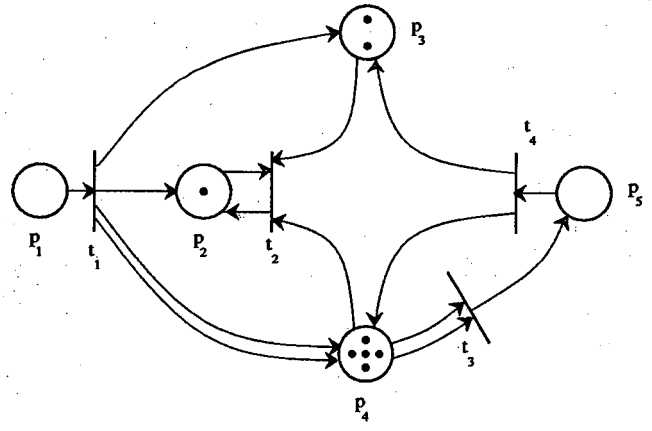
Stochastic Petri nets are useful tools for analyzing complex systems because they are isomorphic with homogeneous Markov processes but have all the expressive capabilities of the original Petri nets [Mol81]. The isomorphic properties of a stochastic Petri net and a Markov process can be seen with the help of the following example. In this example, the simple Petri net shown in Fig. 9.5 will be converted into an equivalent Markov chain. The first step in the



(a)



(b)



(c)

FIGURE 9.3 This figure shows the execution of the Petri net from Fig. 9.1.

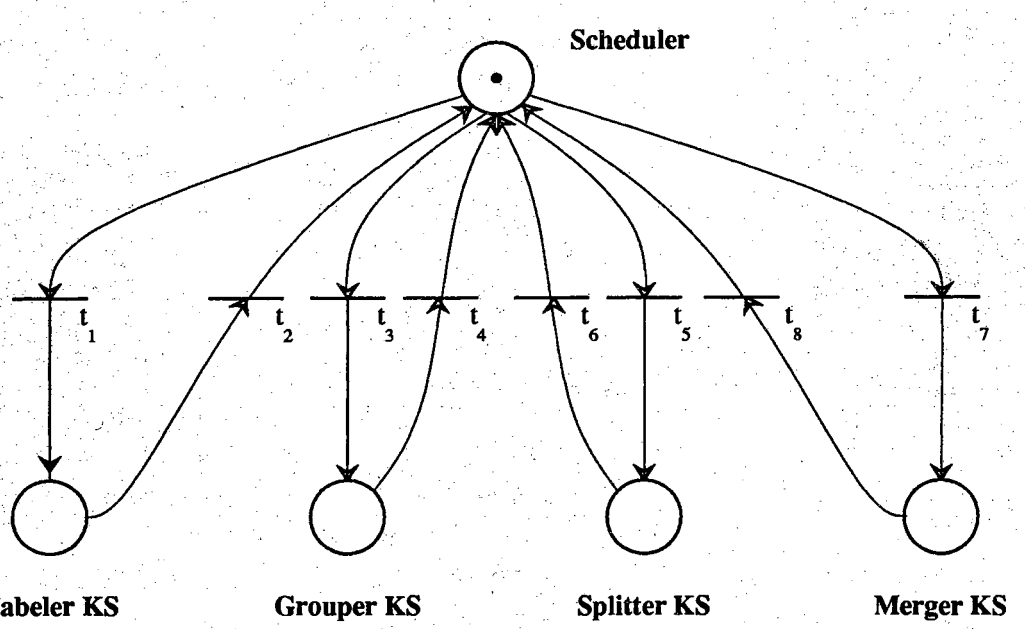


FIGURE 9.4 This figure shows a simple Petri net that can be used to model PSEIKI's control flow.

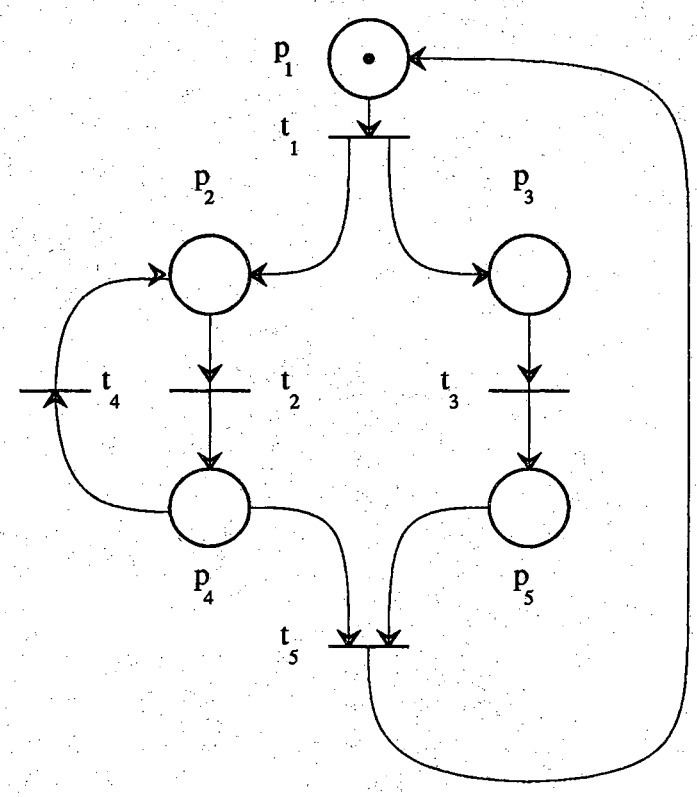


FIGURE 9.5 This figure shows a simple Petri net that is used in the textual explanation of the isomorphism between stochastic Petri nets and Markov processes.

conversion process is the determination of the reachability set of the net given an initial marking. The reachability set of the example Petri net is given in table 9.1. Each row in this table represent a distinct state of the net. The entries in the table represent the number of tokens in a place for a given state.

	p_1	p_2	p_3	p_4	p_5
μ_1	1	0	0	0	0
μ_2	0	1	1	0	0
μ_3	0	0	1	1	0
μ_4	0	1	0	0	1
μ_5	0	0	0	1	1

TABLE 9.1 This table shows the reachability set of the Petri Net shown in Fig. 9.5.

If the mean firing times of the transitions in the stochastic Petri net shown are $\lambda_1 = 2$, $\lambda_2 = 1$, $\lambda_3 = 1$, $\lambda_4 = 3$, $\lambda_5 = 2$, then the following procedure can be used to map the state-space of the net to a Markov chain. A state in the chain is created for every distinct marking in the net. A state-transition is created between two states in the chain if the firing of a single transition in the Petri net will transform the marking of the net from the first state to the second. The mean transition time of the state-transition is set to the mean firing time of the transition that must fire to transform the state of the net from the first state to the second state. For example, marking μ_2 will be transformed into marking μ_4 if transition t_3 fires; thus, in the Markov chain, there is a state-transition from state μ_2 to μ_4 with an average transition time of 1 second, the mean firing time of transition t_3. Fig. 9.6 shows a Markov-chain that is isomorphic to the net shown in Fig. 9.5. In this figure, the mean transition times between states of the chain are indicated by the numbers shown above the state-transitions. The numbers shown below the state-transitions are the *transition-probabilities* of the chain.

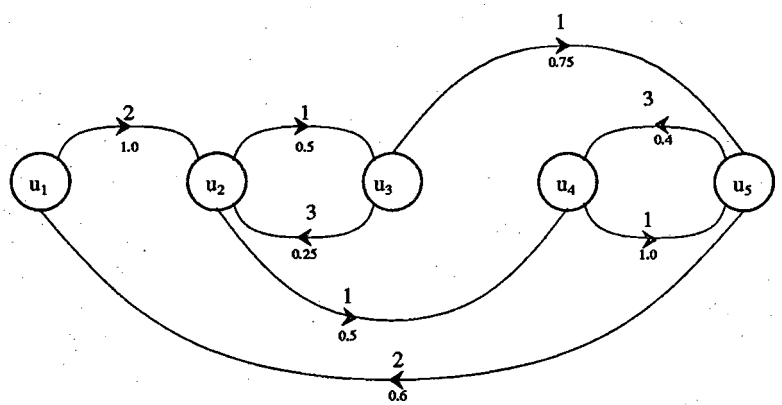


FIGURE 9.6 This figure shows the Markov equivalent to the stochastic Petr net shown in Fig. 9.5.

Once an equivalent Markov chain is constructed from a stochastic Petri net, classical queuing theory techniques [Tri82] may be used to determine the performance of the system by analyzing the chain. For example, the throughput of a system can be estimated by determining the average amount of time that the system needs to transform from a starting state to an ending state and then reset back to the starting state. Queuing theory techniques also can be used to determine the the *steady-state marking probabilities* of the system (the probability that the net will have a particular marking at a given time) by determining the equivalent chain's *limiting state probabilities*. By finding the limiting state probabilities of the Markov-chain in Fig. 9.6, the steady-state marking probabilities of the net in Fig. 9.5 can be shown to be

$$P[\mu_1] = 0.1163$$

$$P[\mu_2] = 0.1860$$

$$P[\mu_3] = 0.0465$$

$$P[\mu_4] = 0.5349$$

$$P[\mu_5] = 0.1163$$

In their current state of development, stochastic Petri nets have a number of drawbacks that limit their use for modeling blackboard systems. First, the reachability set of the net depends on the initial marking. Thus if tokens are used to represent data elements on the blackboard or other problem dependent information, then a new analysis is needed for each problem instantiation. Second, the current formulation of stochastic Petri Nets requires that **every** transition have an exponentially distributed firing time. When modeling complex systems, such as blackboards, it may be neccessary to model transitions that fire immediately on enabeling, require a fixed amount of time to fire, or fire in an amount of time that is a function of the net marking. In addition to these limitations, a final drawback prohibits the use of stochastic Petri Nets for modeling large-scale systems. In general, the size of a Petri net's reachability set will grow exponentially as the number of tokens, places, or transitions in the net increases. Since most queuing theory techniques require the determination of the eigenvalues and eigenvectors of an $N \times N$ matrix when solving a Markov-chain with N states; the problem quickly becomes intractable as the problem size increases. Although stochastic Petri nets cannot currently model systems as complex as blackboards, most researchers are optimistic about the prospect of extending them to handle such large-scale systems. See [RamHo80], [MarCon84], [Zub85], [DugBob85] for some recent work on extended stochastic Petri nets.

CHAPTER 10

EXPERIMENTAL RESULTS

PSEIKI was run on a number of images typical of what would be seen by a sidewalk-navigating mobile robot with downward-slanted cameras. Figs. 10.1 - 10.4 show the results for one such run; Fig. 10.1 shows the edges representing the expected scene and Fig. 10.2 the actual image. Note that the expected scene and the observed image are significantly misregistered. Two of the major edges in the expected scene, in the lower left, are missing entirely in the observed image. The reader should also note the presence of shadow edges in Fig. 10.2. The output of the edge-based preprocessor described in chapter 3 is shown in Fig. 10.3.

The final result produced by PSEIKI consists of labels with associated belief values attached to entities at the edge level and higher levels on the data panel on the blackboard. For example, in Figs. 10.1 - 10.3, if the element at the scene level (the highest blackboard level) with maximum belief is selected and its component edges are displayed, Fig. 10.4 results. This figure shows the edges, their labels, and associated belief values for the scene interpretation that PSEIKI found most believable. In line with the earlier chapters, the percentage value associated with a label indicates PSEIKI's belief in the correctness of the label. For example, PSEIKI has a belief of 0.53 that the lower right edge can be matched with the right-bottom edge of the expected scene. This amount of belief indicates that, at a belief level of 0.47, PSEIKI believes that the edges were mismatched or that the system is ignorant about the validity of the match made.

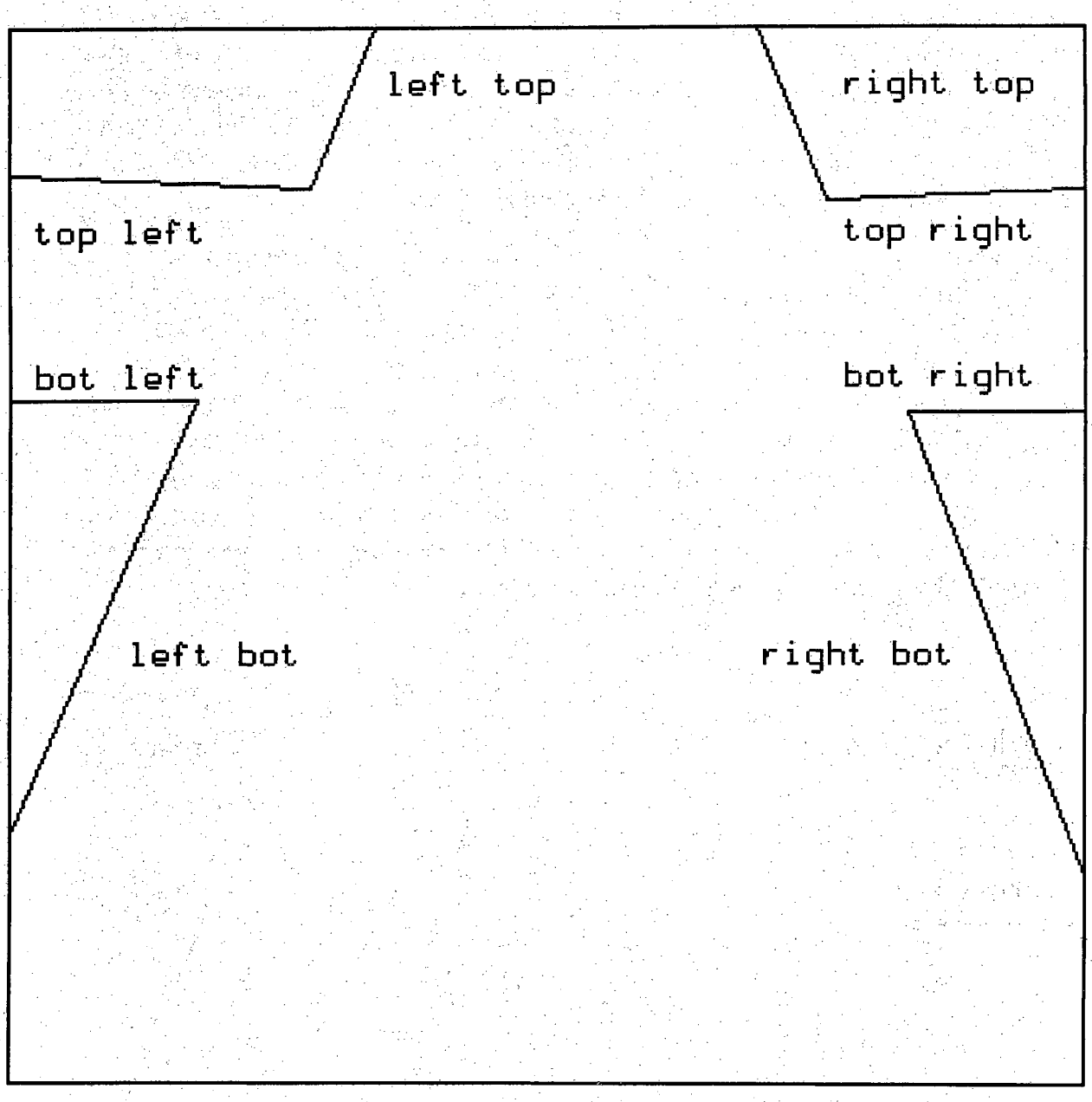


FIGURE 10.1 This figure shows a line drawing of the expected scene with edges labeled.

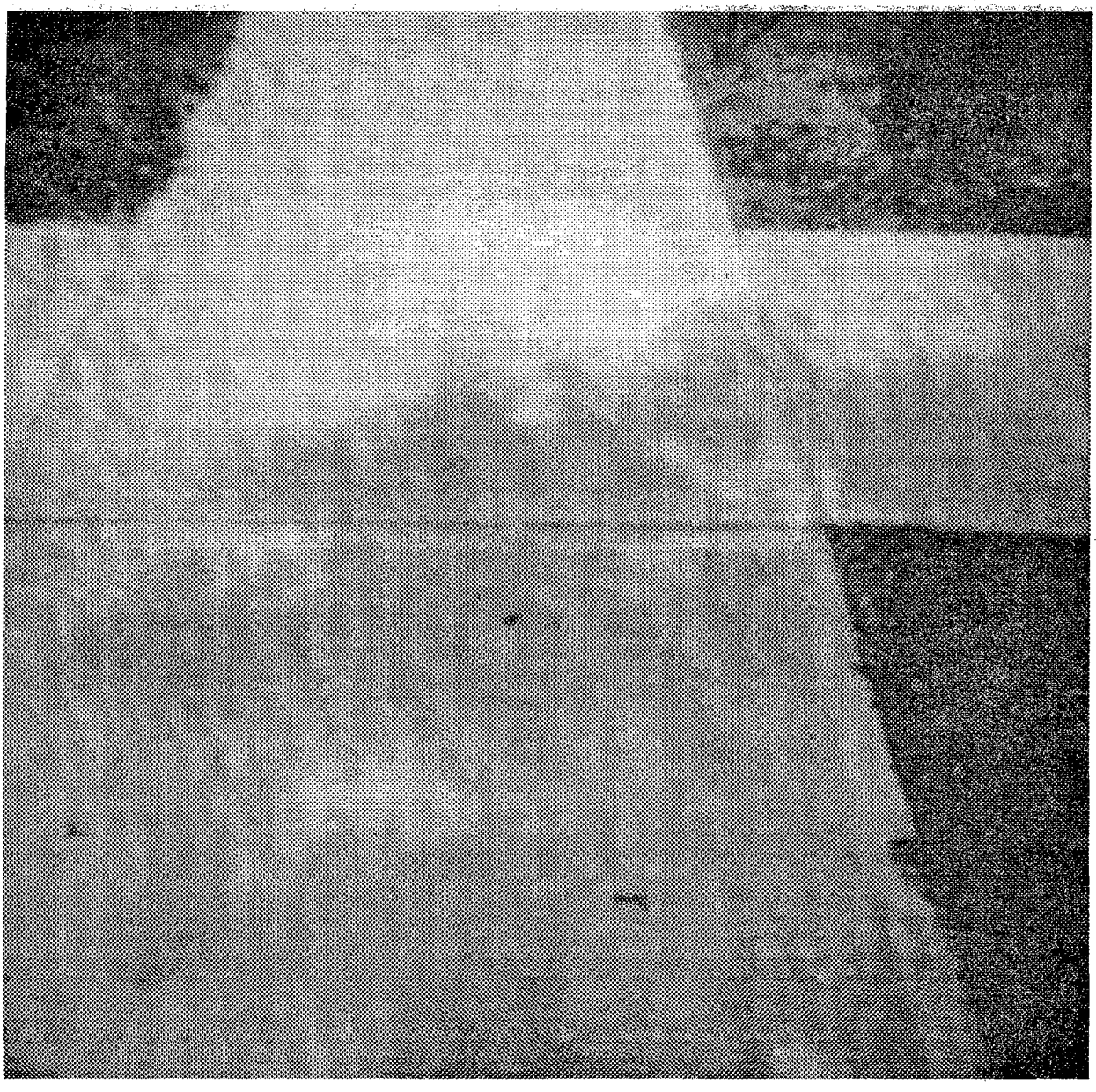


FIGURE 10.2 A sidewalk image used for illustrating PSEIKI's processing is shown here.

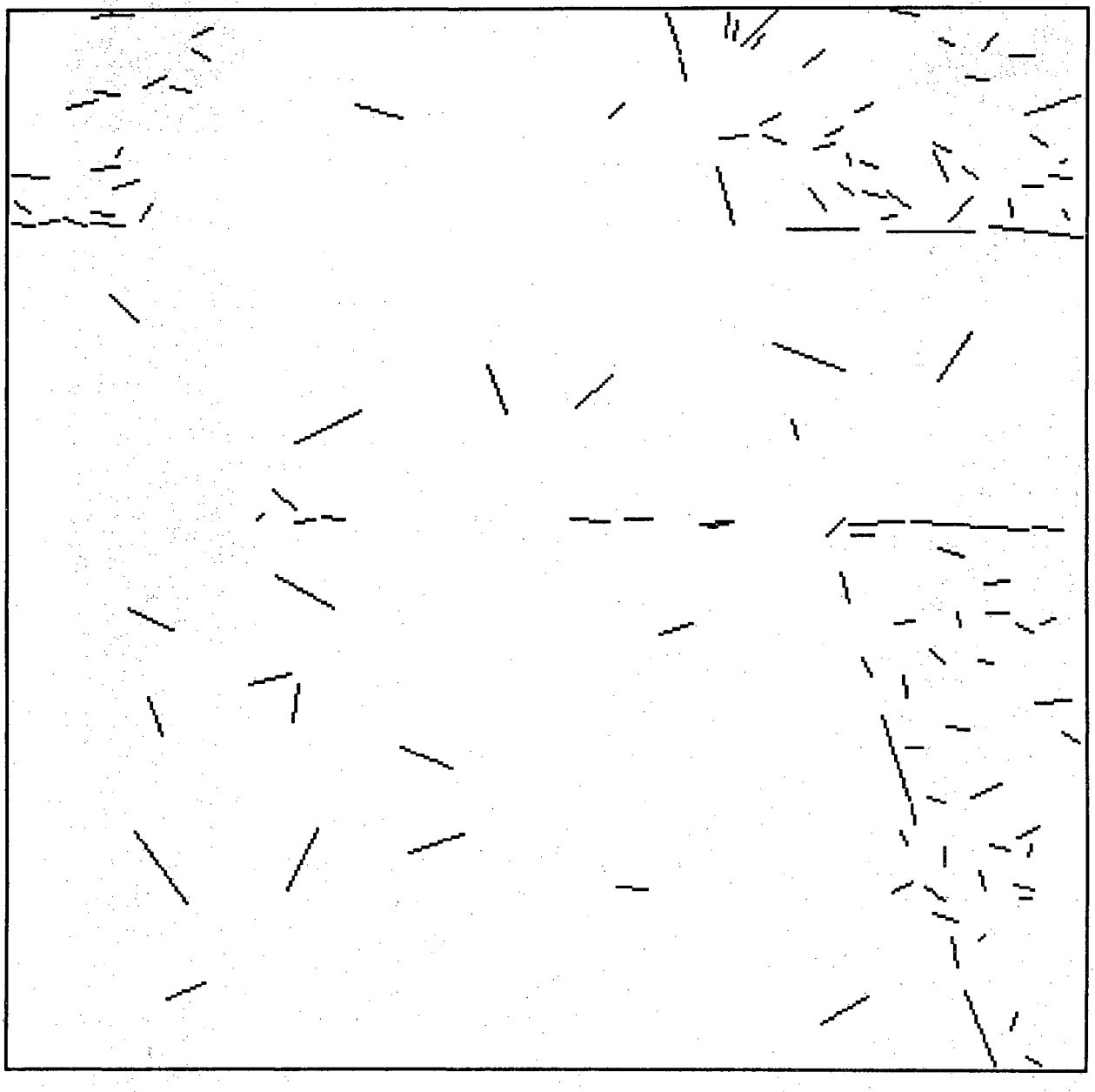


FIGURE 10.3 This figure shows the input to PSEIKI from the preprocessor.

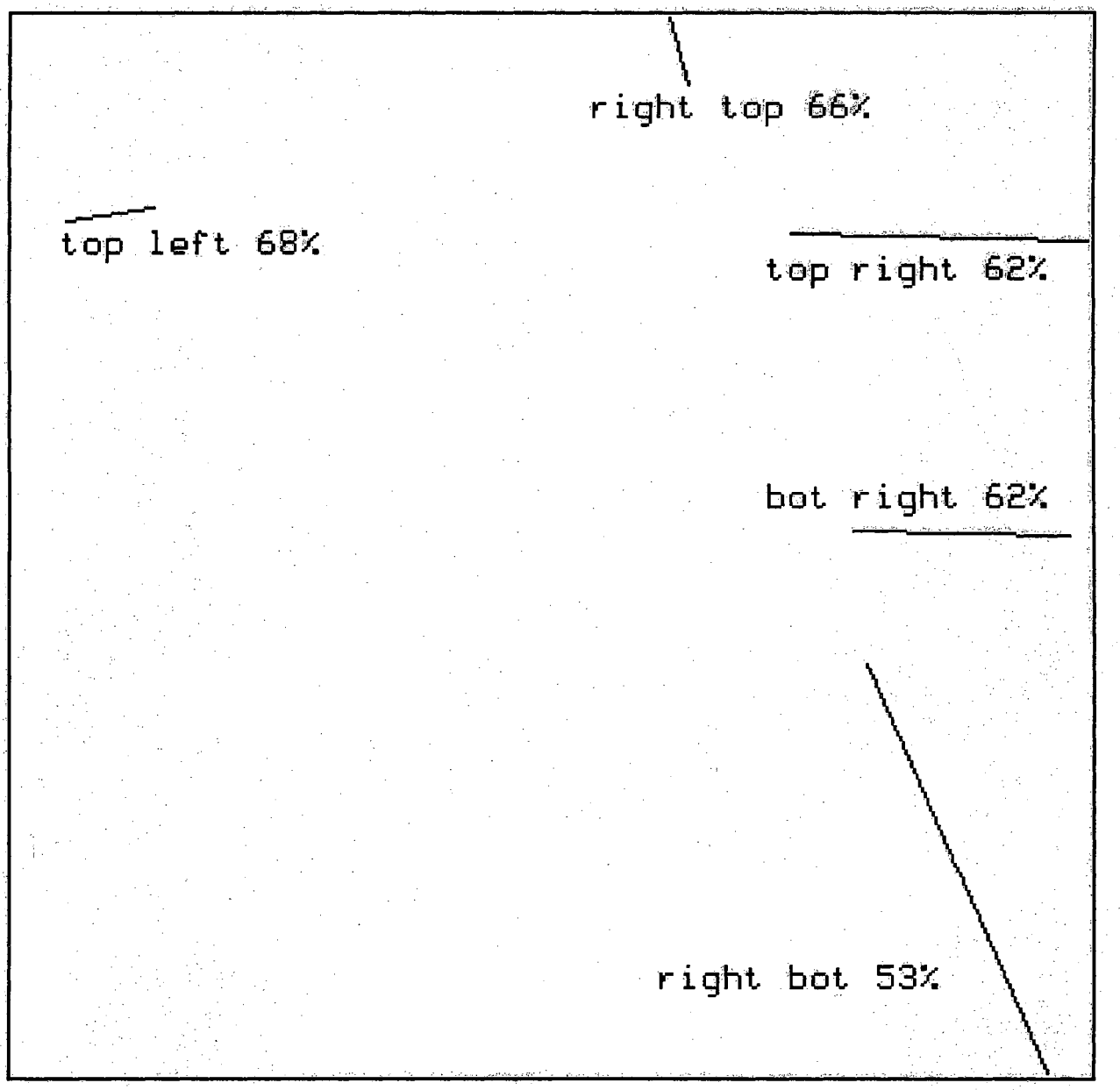


FIGURE 10.4 The output of PSEIKI with detected edges, their labels and associated belief values are displayed here.

APPENDIX A

A BRIEF REVIEW OF DEMPSTER-SHAFER THEORY

In this appendix, we will present a short review of some relevant terminology from the Dempster-Shafer (D-S) theory of evidence accumulation. For a detailed presentation of the theory, the reader is referred to Shafer [Sha76].

In a random experiment, the *frame of discernment* (FOD), Θ, is the set of all possible outcomes. For example, if we roll a die, Θ is equal to the set of possibilities, "the number showing is i," where $1 \leq i \leq 6$; therefore, Θ may be set equal to the set $\{1, 2, 3, 4, 5, 6\}$. The $2^{|\Theta|}$ subsets of Θ are called propositions and the set of all the propositions is denoted by 2^{Θ}. In the die example, the proposition "the number showing is even" would be represented by the set $\{2, 4, 6\}$.

In the D-S theory, *probability masses* are assigned to propositions, meaning to some of the sets in 2^{Θ}, and therein lies a major departure of this theory from the Bayesian formalism in which probability masses must be assigned to the individual elements of Θ. These probability masses must add up to one, and the probability mass assigned to Θ represents ignorance. The interpretation to be given to the probability mass assigned to a subset of Θ is that the mass is free to move to any element of the subset; this interpretation being in consonance with the probability mass assigned to Θ representing ignorance, since this mass may move to any element of the entire FOD. When a source of evidence assigns probability masses to the propositions discerned by Θ, the resulting function is called a *basic probability assignment* (bpa). Formally, a bpa is function $m:2^{\Theta} \rightarrow [0,1]$ where

$$0.0 \leq m(\cdot) \leq 1.0, \quad m(\varnothing) = 0 \quad \text{and} \quad \sum_{X \subseteq \Theta} m(X) = 1.0$$

A belief function, Bel(X), over Θ is defined by

$$\text{Bel}(X) = \sum_{Y \subseteq X} m(Y)$$

In other words, our belief in a proposition X is the sum of probability masses assigned to all the propositions implied by X. *Dempster's rule of combination*, also known as *Dempster's orthogonal sum*, states that given two bpa's, $m_1(\cdot)$ and $m_2(\cdot)$, corresponding to two independent sources of evidence, we may combine them to yield a new bpa $m(\cdot)$ via

$$m(X) = m_1 \oplus m_2 = K \sum_{\substack{X_1 \, X_2 \\ X_1 \cap X_2 = X}} \sum m_1(X_1) m_2(X_2) \quad \text{where}$$

$$K^{-1} = 1 - \sum_{\substack{X_1 \, X_2 \\ X_1 \cap X_2 = \varnothing}} \sum m_1(X_1) m_2(X_2)$$

APPENDIX B

CONVERSION OF CONFIDENCE VALUES TO BASIC PROBABILITY ASSIGNMENTS

Many systems face the problem of converting raw evidence to a form that is usable by the Dempster-Shafer theory of evidence. Garvey, et. al. were the first to investigate the process of converting raw evidence, such as image feature values, into belief functions [GarLow81]; other work on the conversion of sensor readings to belief functions can be found in [LehRey86], [ReyStr86] and [SafGot87]. In this appendix, a scheme to convert confidence values into a bpa is described. In this scheme a confidence value for any subset of an element's FOD is required to be a value between 0.0 and 1.0. A confidence value of 1.0 for a subset of Θ indicates that the evidence source has conclusive evidence that the element's identity is in that subset. Conversely, a confidence value of 0.0 indicates a lack of evidence that the element's identity is in the subset. To formalize the notion of confidence values, a **confidence function**, Conf, is defined.

$$\text{Conf}: 2^{\Theta} \rightarrow [0, 1]$$

The idea is that the value of this function for any subset represents the amount of evidence provided by a source suggesting that the element's identity is in the subset. Note that this notion is related to the concept of a probability mass in a basic probability assignment; however, a bpa has other properties that are not required of a confidence function. Although a confidence function may not have all the necessary properties of a bpa, a bpa can be defined in terms of an underlying confidence function. To define a bpa, $m(\cdot)$, in terms of a confidence function, it must be defined so that it satisfies three properties.

1) $0.0 \leq m(\cdot) \leq 1.0$
   The confidence function meets this criteria by definition.

2) $m(\varnothing) = 0.0$
   This property is obtained by setting the probability mass of the null set to zero. This action makes intuitive sense because the null set represents the case in which the element's identity is not a member of the FOD. If this were the case, the FOD would be incomplete and a new, more complete one would be needed.

3) $\sum_{\psi \subseteq \Theta} m(\psi) = 1.0$
   This requirement states that the evidence source generating the bpa has unity total-belief. When forming a bpa with this property, the concept of the source's *total confidence* is helpful. A source's total confidence is defined to be

$$\text{Conf}_{tot} = \sum_{\substack{\psi \subseteq \Theta \\ \psi \neq \varnothing}} \text{Conf}(\psi)$$

   This concept can be used to break the problem into three cases.

1) $Conf_{tot} = 1.0$
In this case, the confidence function is a bpa. Therefore, define $m(x) = Conf(x)$ for all $x \in 2^{\Theta}$, $x \neq \varnothing$.

2) $Conf_{tot} < 1.0$
In this case, Conf incompletely specifies the source's belief. A bpa can be defined by assigning the uncommitted portion of the source's belief, its ignorance about the identity of the element, to the entire FOD, Θ.

$$m(x) = \begin{cases} 1.0 - Conf_{tot} & x = \Theta \\ 0.0 & x = \varnothing \\ Conf(x) & \text{else} \end{cases}$$

3) $Conf_{tot} > 1.0$
In this case, the evidence source has over-specified its belief. A bpa is defined by normalizing all the confidence values by its total confidence.

$$m(x) = \frac{Conf(x)}{Conf_{tot}} \quad \text{for all } x \in 2^{\Theta},\ x \neq \varnothing$$

After the preceding operations are applied to the confidence function, a bpa for the evidence source, $m(\cdot)$, results. Note that defining the bpa in this manner does not affect the validity of the first two requirements for a bpa; this is apparent because $Conf_{tot} \geq Conf(\cdot) \geq 0$.

To see more clearly how the conversion process works, consider the following example. Assume for this example that an evidence source is being used to determine the identity of an object with FOD $\Theta = \{\theta_A, \theta_B, \theta_C, \theta_D\}$. If the evidence source provides non-zero weights only to members of Θ, then the following confidence function might result*:

$$Conf(\theta_A) = 0.7$$
$$Conf(\theta_B) = 0.1$$
$$Conf(\theta_C) = 0.4$$
$$Conf(\theta_D) = 0.05$$

If the total confidence exceeds unity, as in this example, the confidence values are normalized by the summed value resulting in the following bpa over Θ:

* Note that, in general, an evidence source could provide values to any element of 2^{Θ}, not just elements of Θ.

$m(\theta_A) = 0.56$

$m(\theta_B) = 0.08$

$m(\theta_C) = 0.32$

$m(\theta_D) = 0.04$

$m(\cdot) = 0.0$ for all other subsets of Θ

On the other hand, the evidence source could have produced values that sum to less than one, as in the following case:

$Conf(\theta_A) = 0.7$

$Conf(\theta_B) = 0.1$

$Conf(\theta_C) = 0.0$

$Conf(\theta_D) = 0.05$

Since the measures now sum to less than unity, there is no reason to normalize. Instead, they are converted directly into a bpa in the following manner:

$m(\theta_A) = 0.7$

$m(\theta_B) = 0.1$

$m(\theta_C) = 0.0$

$m(\theta_D) = 0.05$

$m(\Theta) = 0.15$

$m(\cdot) = 0.0$ for all other subsets of Θ

Note that the amount of belief assigned to Θ is equal to 0.15; this is the difference between unity belief and the evidence source's total confidence. Setting the probability mass in Θ to the difference seems intuitively correct for the simple reason that $Conf(\theta_i)$ is a good measure of the confidence that the object's identity is θ_i. Clearly if the object is not thought to correspond to any of the elements in its FOD to a sufficiently high degree, then some belief may be uncommitted. In the above assignment, $m(\Theta) = 0.15$ represents the uncommitted portion of the belief.

ACKNOWLEDGEMENTS

We would like to thank Dr. Lynn Garn, Team Chief of the Image Understanding and Artificial Intelligence Group, and Mr. Tim Williams, both of the Army Center for Night Vision and Electro-Optics, for many fruitful discussions that educated us on the limitations of the existing methodologies for automatic target recognition. Furthermore, the work described in this report would not have been possible without the help of many people at Purdue's Robot Vision Lab. Specifically, we would like to thank Tom Underwood and Yura Kim for their input on the low level processing needed by PSEIKI. We would also like to thank Richard Coe for his work on the design and development of PSEIKI's debugging software. Jeffery Lewis also played a large role in the development of PSEIKI's software; his development of a general image processing environment was much help. Finally, Matt Carroll played a large role in the evolution of PSEIKI with his help in data collection.

BIBLIOGRAPHY

[AhoHop74] Aho, A. V., J. E. Hopcraft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, Mass., 1974.

[AndKak87] Andress, K. M. and A. C. Kak, "A Production System Environment for Integrating Knowledge with Vision Data," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 1-12, Morgan-Kaufmann Publishers, Inc., 1987.

[AndKak88] Andress, K. M. and A. C. Kak, "Evidence Accumulation and Flow of Control in a Hierarchical Spatial Reasoning System," *The AI Magazine*, vol. 9, no. 2, pp. 75-95, 1988.

[BakBin81] Baker, H. H. and T. O. Binford, "Depth from Edge and Intensity based Stereo," *Proceedings IJCAI*, vol. 6, pp. 631-636, 1981.

[BalBro82] Ballard, D. H. and C. M. Brown, *Computer Vision,* Prentice Hall, Englewood Cliffs, New Jersey, 1982.

[BarTho81] Barnard, S. T. and W. B. Thompson, "Disparity Analysis of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 333-340, 1981.

[Bar83] Barnard, S. T., "Interpreting Perspective Images," *Artificial Intelligence*, vol. 21, pp. 435-462, 1983.

[Bar81] Barnett, J. A., "Computational Methods for a Mathematical Theory of Evidence," *Proceedings IJCAI*, pp. 868-875, 1981.

[BarTen81] Barrow, H. G. and J. M. Tenenbaum, "Interpreting Line Drawings as Three-Dimensional Surfaces," *Artificial Intelligence*, vol. 17, pp. 75-116, 1981.

[BesJai85] Besl, P. J. and R. C. Jain, "Three-Dimensional Object Recognition," *ACM Computing Surveys*, vol. 17, no. 1, pp. 75-144, 1985.

[Bes88] Besl, P. J., "Geometric Modeling and Computer Vision," *Proceedings of the IEEE*, vol. 76, no. 8, pp. 936-958, 1988.

[BinLev87] Binford, T. O., T. S. Levitt, and W. B. Mann, "Bayesian Inference in Model-Based Machine Vision," *Proceedings AAAI Workshop on Uncertainty in Artificial Intelligence*, Seattle, July 1987.

[BriFen70] Brice, C. R. and C. L. Fennema, "Scene Analysis Using Regions," *Artificial Intelligence*, vol. 1, no. 3, pp. 205-226, 1970.

[Bro81] Brooks, R. A., "Symbolic Reasoning Among 3-D Models and 2-D Images," *Artificial Intelligence*, vol. 17, pp. 285-348, 1981.

[BroFar85] Brownston, L., R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5*, Addison-Wesley, 1985.

[ConMun87] Connolly, C. I., J. L. Mundy, J. R. Stenstrom, and D. W. Thompson, "Matching from 3-D Range Models into 2-D intensity Scenes," *Proceedings on 1st International Conference on Computer Vision*, pp. 65-72, 1987.

[CorGal87] Corkill, D. D., K. Q. Gallagher, and P. M. Johnson, "Achieving Flexibility, Efficiency, and Generality in Blackboard Architectures," *Proceedings AAAI*, 1987.

[CroSan87] Crowley, J. L. and A. C. Sanderson, "Multiple Resolution Representation and Probabilistic Matching of 2-D Grey-Scale Shape," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 1, pp. 113-121, 1987.

[CroRam87] Crowley, J. L. and F. Ramparany, "Mathematical Tools for Representing Uncertainty in Perception," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 293-302, Morgan-Kaufmann Publishers, Inc., 1987.

[DavHwa85] Davis, L. S. and S. S. V. Hwang, "The SIGMA Image Understanding System," *Proceedings of the IEEE Workshop on Computer Vision, Representation and Control*, pp. 19-26, 1985.

[DubPra85] Dubois, D. and H. Prade, "Combination and Propagation of Uncertainty with Belief Functions," *Proceedings IJCAI*, 1985.

[DubPra86] Dubois, D. and H. Prade, "Set-theoretic Operations on Bodies of Disjunctive or Conjunctive Evidence," *Proceedings of the North American Fuzzy Information Processing Society*, pp. 107-124, New Orleans, 1986.

[DudHar73] Duda, R. O. and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.

[DugBob85] Dugan, J. B., A. Bobbio, G. Ciardo, and K. Trivedi, "The Design of a Unified Package for the Solution of Stochastic Petri Net Models," *Proceedings of International Conference on Timed Petri Nets*, 1985.

[DurLes87] Durfee, E. H., V. R. Lesser, and D. D. Corkill, "Coherent Cooperation Amoung Communicating Problem Solvers," *IEEE Transactions on Computers*, vol. Vol C-36, no. 11, pp. 1275-1291, 1987.

[Ebe76] Eberlein, R. B., "An iterative Gradient Edge Detection Algorithm," *Computer Graphics and Image Processing*, vol. 5, pp. 245-253, Acedemic Press, 1976.

[ErmHay80] Erman, L. D., F. Hayes-Roth, V. R. Lesser, and D. R. Reddy, "The Hearsay-II Speech Understanding System: Integrating Knowledge to Resolve Uncertainty," *ACM Computing Surveys*, pp. 213-253, 1980.

[ErmLon81] Erman, L. D., P. E. London, and S. F. Fickas, "The Design and an Example Use of Hearsay-III," *Proceedings Joint Int'l Conference on Artificial Intelligence*, 1981.

[FauPra79] Faux, I. D. and M. J. Pratt, *Computational Geometry for Design and Manufacturing*, Ellis Horwood, Ltd., New York, 1979.

[FenLes77] Fennell, R. D. and V. R. Lesser, "Parallelism in Artificial Intelligence Problem Solving: A Case Study of Hearsay II," *IEEE Transactions on Computers*, vol. Vol C-26, no. 2, pp. 98-111, 1977.

[FolVan82] Foley, J. D. and A. VanDam, *Fundamentals of Interactive Computer Graphics*, Addison Wesley, Reading, Mass., 1982.

[For86] Forgy, C. L., *OPS/83 User's Manual and Report*, Production Systems Technologies, Inc., 1986.

[GarJoh79] Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, 1979.

[GarCor87] Garvey, A., C. Cornelius, and B. Hayes-Roth, "Computational Costs versus Benefits of Control Reasoning," *Proceedings AAAI*, 1987.

[GarLow81] Garvey, T. D., J. D. Lowrance, and M. A. Fischler, "An Inference Technique for Integrating Knowledge from Disparate Sources," *Proceedings IJCAI*, pp. 319-325, 1981.

[GorSho85] Gordon, J. and E. H. Shortliffe, "A Method for Managing Evidential Reasoning in a Hierarchical Hypothesis Space," *Artificial Intelligence*, vol. 26, pp. 323-357, 1985.

[Gri81a] Grimson, W. E. L., "A computer implementation of a theory of human stereo vision," *Phil. Trans. Roy. Soc. Lon.*, vol. B 292, pp. 217-253, 1981.

[Gri81b] Grimson, W. E. L., *From Images to Surfaces: A Computational Study of the Human Early Visual System*, MIT Press, 1981.

[HanRis78] Hanson, A. R. and E. M. Riseman, "VISIONS: A Computer System for Interpreting Scenes," in *Computer Vision Systems*, ed. E. M. Riseman, pp. 303-333, Academic Press, 1978.

[HarSha85] Haralick, R. M. and L. G. Shapiro, "SURVEY: Image Segmentation Techniques," *Computer Vision, Graphics, and Image Processing*, vol. 29, pp. 100-132, Academic Press, 1985.

[HarMar85] Hartquist, E. E. and H. A. Marisa, "PADL-2 Users Manual," Production Automation Project, The University of Rochester, UM-10/2.1, 1985.

[Hay85] Hayes-Roth, B., "A Blackboard Architecture for Control," *Artificial Intelligence*, pp. 251-321, 1985.

[HayLes77] Hayes-Roth, F. and V. R. Lesser, "Focus of Attention in the Hearsay-II Speech Understanding System," *Proceedings IJCAI*, 1977.

[HofJai87] Hoffman, R. and A. K. Jain, "Segmentation and Classification of Range Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-9, no. 5, pp. 608-620, 1987.

[HorPav76] Horowitz, S. L. and T. Pavlidis, "Picture Segmentation by a Tree Traversal Algorithm," *Journal of the ACM*, vol. 23, no. 2, pp. 368-388, April 1976.

[HuSto87] Hu, Gongzhu and George Stockman, "3-D Scene Analysis Via Fusion of Light Striped Image and Intensity Image," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 138-147, Morgan-Kaufmann Publishers, Inc., 1987.

[HunJay87] Huntsberger, T. L. and S. N. Jayaramamurthy, "A Framework for Multi-Sensor Fusion in the Presence of Uncertainty," *Proceedings of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Fusion*, Morgan-Kaufmann Publishers, Inc., 1987.

[JohHay87] Johnson, M. V. Jr. and B. Hayes-Roth, "Intergrating Diverse Reasoning Methods in the BB1 Blackboard Control Architecture," *Proceedings AAAI*, 1987.

[KakRob87] Kak, A. C., B. A. Roberts, K. M. Andress, and R. L. Cromwell, "Experiments in the Integration of World Knowledge with Sensory Information for Mobile Robots," *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, pp. 734-741, 1987.

[Kim88] Kim, W. Y. and A. C. Kak, "A Cross Scanning Structured Light System for 3-D Robot Vision," School of Electrical Engineering, Purdue University, Technical Report (in preparation), 1988.

[Koh47] Kohler, W., *Gestalt Pshychology,* Liverright, Ney York, 1947.

[Kui77] Kuipers, B., "Representing Knowledge of Large-Scale Space," MIT AI Lab Technical Report AI-TR-418, July 1977.

[Kyb87] Kyburg, H. E. Jr., "Bayesian and Non-Bayesian Evidential Updating," *Artificial Intelligence*, pp. 271-293, 1987.

[LawMcC87] Lawton, D. T. and C. C. McConnell, "Perceptual Organization Using Interestingness," *Proc. of the 1987 AAAI Workshop on Spatial Reasoning and Multi-Sensor Integration*, pp. 405-419, Morgan-Kaufmann Publishers, Inc., 1987.

[LehRey86] Lehrer, N. B., G. Reynolds, and J. Griffith, "A Method for Initial Hypothesis Formation in Image Understanding," *Proc. of the 1st IEEE Conference on Computer Vision*, 1986.

[LesErm77] Lesser, V. R. and L. D. Erman, "A retrospective View of the Hearsay-II Architecture," *Proceedings Joint Int'l Conference on Artificial Intelligence*, 1977.

[LesErm80] Lesser, V. R. and L. D. Erman, "Distributed Interpretation: A Model and Experiment," *IEEE Transactions on Computers*, vol. Vol C-29, no. 12, pp. 1144-1289, 1980.

[LevLaw87] Levitt, T. S., D. T. Lawton, D. M. Chelberg, and P. C. Nelson, "Qualitative Navigation," *Proc. of DARPA Image Understanding Workshop*, Los Angeles, Febuary 1987.

[Law85] Lowe, D. G., *Perceptual Organization and Visual Recognition,* Kluwer Academic , Boston, 1985.

[MarPog79] Marr, D. and T Poggio, "A Theory of Human Stereo Vision," *Proc. R. Soc. Lond.*, vol. B, no. 204, pp. 301-328, 1979.

[Mar82] Marr, D., *Vision,* W. H. Freeman and Co., 1982.

[MarCon84] Marsan, M. A., G. Conte, and G. Balbo, "A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems," *ACM Transactions on Computer Systems*, vol. 2, no. 2, 1984.

[Mas87] Mashburn, T., "A Polygonal Solid Modeling Package," M.S. Thesis, School of Mechanical Engineering, Purdue University, 1987.

[MatHwa85] Matsuyama, T. and S. S. V. Hwang, "SIGMA: a Framework for Image Understanding - Integration of Bottom-up and Top-down Analysis," *Proceedings IJCAI*, pp. 908-915, 1985.

[MckHar85] McKeown, D. M. Jr., W. A. Harvey, Jr., and J. McDermott, "Rule-Based Interpretation of Aerial Imagery," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-7, no. 5, pp. 570-585, 1985.

[MedNev84] Medioni, G. and R. Nevatia, "Matching Images Using Linear Features," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, pp. 675-685, 1984.

[MitHar87] Mitchell, D. H., S. A. Harp, and D. K. Simkin, "A Knowledge Engineer's Comparison of Three Evidence Aggregation Methods," *Proceedings of Uncertainty in Artificial Intelligence Workshop*, pp. 297-312, 1987.

[Mol82] Molloy, M. K., "Performance Analysis Using Stochatic Petri Nets," *IEEE Transactions on Computers*, vol. C-31, no. 9, 1982.

[Mor85] Mortenson, M. E., *Geometric Modeling,* John Wiley & Sons, New York, 1985.

[Mor66] Morton, G. M., "A computer oriented geodetic data base and a new technique in file sequencing," unpublished, IBM, Ottawa, Canada, 1966.

[NagMat80] Nagao, M. and T. Matsuyama, *A Structural Analysis of Complex Aerial Photographs,* Plenum Press, 1980.

[NazLev84] Nazif, A. M. and M. D. Levine, "Low Level Segmentation: An Expert System," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-6, no. 5, pp. 555-577, 1984.

[NavBab80] Nevatia, R. and K. R. Babu, "Linear Feature Extraction and Description," *Computer Graphics and Image Processing*, vol. 13, pp. 257-269, 1980.

[NiiFei78] Nii, H. P. and E. A. Feigenbaum, "Rule-based Understanding of Signals," in *Pattern Directed Inference Systems*, ed. F. Hayes-Roth, pp. 53-68, Academic Press, 1978.

[Nii86b] Nii, P. H., "Blackboard Systems from a Knowledge Engineering Perspective," *The AI Magazine*, pp. 82-106, August 1986.

[Nii86a] Nii, P. H., "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of the Blackboard Architectures," *The AI Magazine*, pp. 38-53, Summer 1986.

[Pea84] Pearl, J., *Heuristics,* Addison Wesley, Reading, Mass, 1984.

[Pea86] Pearl, J., "Fusion, Propagation, and Structuring in Bayesian Networks," *Artificial Intelligence*, 1986.

[Pet81] Peterson, J. L., *Petri Net Theory and the Modeling of Systems,* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[RamHo80] Ramamoorthy, C. V. and G. S. Ho, "Performance Evaluation of Asynchronous Concurrent Systems Using Petri Nets," *IEEE Transactions on Software Engineering*, vol. SE-6, no. 5, 1980.

[ReyStr86] Reynolds, G., D. Strahman, N. Lehrer, and L. Kitchen, "Plausible Reasoning and the Theory of Evidence," COINS Tech. Report 86-11, University of Massachussetts, 1986.

[RosKak82] Rosenfeld, A. and A. C. Kak, *Digital Picture Processing, Vols. 1 & 2,* Academic Press, Orlando, 1982.

[SafGot87] Safranek, R. J., S. Gottschlich, and A. C. Kak, "Evidential Reasoning Using Binary Frames of Discernment for Verification Vision," *Submitted for publication*, 1987.

[Sam84b] Samet, H., "A Tutorial on Quadtree Research," in *Multiresolution Image Processing and Analysis*, Springer-Verlag, 1984.

[Sam84a] Samet, H., "The Quadtree and Related Hierarchical Data Structures," *ACM Computing Surveys*, vol. 16, no. 2, ACM, June 1984.

[Sha76] Shafer, G., *A Mathematical Theory of Evidence*, Princeton University Press, 1976.

[ShaLog87] Shafer, G. and R. Logan, "Implementing Dempster's Rule for Hierarchical Evidence," *Artificial Intelligence*, vol. 33, pp. 271-298, 1987.

[ShaHar81] Shapiro, L. G. and R. M. Haralick, "Structural Descriptions and Inexact Matching," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-3, no. 5, pp. 504-519, 1981.

[Sme76] Smets, P., "Combining Non-Distinct Evidences," *Proceedings North American Fuzzy Information Processing Society*, pp. 544-548, New Orleans, 1986.

[Tri82] Trivedi, K. S., *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Prentice-Hall, Englewood Cliffs, NJ, 1982.

[VoeReq77] Voelcker, H. B. and A. A. G. Requicha, "Geometric Modeling of Mechanical Parts and Processes," *IEEE Computer*, 1977.

[Wat70] Watkins, G. S., "A Real-Time Visible Surface Algorithm," Univeristy of Utah Computer Science Department, UTEC-CSc-70-101, 1970.

[WatArv84] Watson, L. T., K. Arvind, R. W. Ehrich, and R. M. Haralick, "Extraction of Lines and Regions From Grey Tone Line Drawings," *Pattern Recognition*, vol. 17, no. 5, pp. 493-507, 1984.

[WatArv87] Watson, L. T., K. Arvind, R. W. Ehrich, and R. M. Haralick, "Extraction of Lines and Regions From Grey Tone Line Drawing Images," *Pattern Recognition*, vol. 17, no. 5, 1987.

[WitTen83a] Witkin, A. P. and J. M. Tenenbaum, "On the Role of Structure in Vision," in *Human and Machine Vision*, ed. Rosenfeld, pp. 481-543, Academic Press, New York, 1983.

[WitTen83b] Witkin, A. P. and J. M. Tenenbaum, "What is Perceptual Organization For?," *Proceedings IJCAI*, pp. 1023-1026, 1983.

[YanKak86] Yang, H. S. and A. C. Kak, "Determination of the Identity, Position and and Orientation of the Topmost Object in a Pile.," *Computer Vision, Graphics, and Image Processing*, vol. 36, pp. 229-255, Academic Press, 1986.

[Yen86] Yen, J., "A Reasoning Model Based on an Extended Dempster-Shafer Theory," *Proceedings AAAI*, pp. 125-131, 1986.

[Zis78] Zisman, M. D., "Use of Production Systems for Modeling Asynchronous, Concurrent Processes," in *Pattern Directed Inference Systems*, ed. F. Hayes-Roth, pp. 53-68, Academic Press, 1978.

[Zub85] Zuberek, W. M., "Performance Evaluation Using Extended Timed Petri Nets," *Proceedings of International Conference on Timed Petri Nets*, 1985.

[Zuc76] Zucker, S. W., "Region Growing: Childhood and Adolescence," *Computer Graphics and Image Processing*, vol. 5, pp. 382-399, 1976.